アスキー・ラーニングシステム
ASCII Learning System
①入門コース

マイクロコンピュータの基本ソフトウェア

入門CP/M®

村瀬康治——著

アスキー出版局

入門CP/M®

村瀬 康治 著

アスキー出版局

*CP/M Learning System*全3巻の構成

この「CP／M Learning System」全３巻は，次のように構成されています．

入門 CP／M………CP／M がどの様なものであるかを解説し，CP／M を使うための基礎知識，日常よく使う各コマンドの実習などをやさしく具体的に解説します．

今まで CP／M については全く知らなかった読者でも，本書により CP／M の概要を理解し，一通り CP／M が使えるようになれるよう配慮されています．

実習 CP／M………CP／M の全コマンドと，そのほとんどすべての使い方を徹底的かつやさしく，具体的に実習しながら解説します．また，CP／M のハードウェアおよびソフトウェアの構成についても解説し，CP／Mの実用例として，CP／Mアセンブラによるマシン語開発の全過程を実習します．

本書は，読者が本格的に CP／M を使うようになった場合，CP／M のコマンド・ハンドブックとして，随時参照することになるでしょう．

応用 CP／M………CP／M をさらに深く広く応用する場合についての解説書です．マクロ・アセンブラによるマシン語開発，システム・コール，各種高級言語の使用例，アプリケーションやユーティリティの実行なども，分かり易い実行例に基づいて解説します．さらに CP／M の内部構造，BIOS の詳細など，本格的な CP／M ユーザーとして，いずれは必要となる知識を提供します．

各巻はそれぞれにまとまっていますので，必要に応じてどの巻を手にされても利用することができます．

本シリーズは，CP／M version ２．２を基にして書かれていますが，旧バージョンである，version １．４を使う場合のことを考慮し，共通でないコマンドについては，そのつど注意書きを付け加えてあります．

著者まえがき

1981年8月，IBMがCP／Mライクな独自のOSを載せた〝THE IBMパーソナル・コンピュータ〟を低価格で発表し，いよいよ本格的なパーソナル・コンピュータ時代に突入しました．一方，ゼロックスは〝XEROX-820〟を発表，ヒューレット・パッカードもデスクトップ・コンピュータ〝HP125〟を発表し，これらにはいずれもCP／Mが採用されています．9月には，CDC社もCP／Mマシン〝model-110〟を発表し，メインフレーム・メーカーがCP／Mを載せて続々とパーソナル・コンピュータ市場に乗り出して来ました．

このような動きは，これら超有力メーカーも，いわゆる〝流通ソフトウェア〟とそれを利用するために必要な，基本ソフトウェアである CP／M を十分考慮し，評価していることを表わしています．

しかし，アメリカは，このような超有力メーカーが参入する以前，すでに数年前から CP／M 全盛の国であり，マイクロ・コンピュータのソフトウェアは CP／M なしでは全く考えられないと言ってもよいほど CP／M に依存しています．一側面として，アメリカでは Apple II パーソナル・コンピュータ (6502 CPU) 上で，CP／M を走らせるためのオプションである〝Z80ソフトカード〟(6502マシン上で Z80を走らせるためのハードウェアと CP／M ディスケットの組合せ．マイクロソフト社製）の売上げが，高収益を誇るマイクロソフト社のドル箱となっていることからも，いかに CP／M が一般に必要とされているかが分ります．(CP／M のデジタル・リサーチ社とライバル関係にあるマイクロソフト社が相手側の CP／M を使わせるためのオプションを売り，その CP／M 上で走る自社のソフトウェアを売っていることがおもしろい)

国内でも，富士通の MICRO-8 が新たに CP／M マシンとして加わるのを始め，各社の代表的機種はすでにほとんどで CP／M が走っています．そして，まだ CP／M マシンを持たないメーカーは，今，全力でその開発を急いでいることでしょう．

今日，CP／M は8ビットのみならず，16ビットにおいても，マイクロ・コンピュータを本格的に応用するには不可欠の国際的基本ソフトウェアとして，絶対的な位置を占めるに到っているのです．

1981年6月，CP／M の開発者であり，Digital Research 社のプレジデントである Gary Kildall 氏を招いて，CP／M，CP／M-86と，それらのファミリーについてのセミナー★が，ホテル・ニューオータニで開かれました．大手企業のほとんどと，主要システム・ソフトウェア産業のエキスパート達が大勢集まり，大変な盛況でした．素顔の Kildall 氏は，年よりずっと若く，実に気さくな好青年とい

★ セミナー――マイクロソフトウエア・アソシエイツ（Digital Research 極東総代理店）主催．

う感じであり，アスキー出版から発売されている筆者監修の「標準 CP／M ハンドブック」をプレゼントしたところ，東洋の文字で書かれた CP／M の本に，大変な喜びようでした．その時，日本の CP／M ユーザーにと，サインを頂いてありますので，この紙面を借りて掲載させていただきます．

もちろん本書とサインとは全然関係ありません．

本書が，読者と CP／M との出逢いの書となって，大勢の CP／M ユーザーが生まれ，より本格的にマイクロ・コンピュータを利用するための案内役をはたせれば，と願っています．

1981年 9 月　　村瀬康治

Best Regards to
my cordial Japanese
Friends
Gary A. Kildall

目次

実習一覧

CP／Mの歴史

ＣＰ／Ｍは，1973年に当時インテル社のコンサルタントとしてＰＬ／Ｍ★を開発していたGary Kildallによって，ＤＥＣの大型マシンＴＯＰＳ-10をモデルにその原形が作られました．しかし，インテル社を始め，当時の有力メーカーはほとんど興味を示さず，Kildallは大変失望しました．最初のVersion は1.3として1974年に世に出ましたが反応は少なく，世に広まり始めたのは，1975年にＩＭＳＡＩのシステムで採用されてからでした．その後,Versionは1.4にＵＰされ，1976年KildallはDigital Research社を設立し，ＣＰ／Ｍは爆発的に普及して行ったのです．

1979年には，大容量のハードディスクにも対応できるように，Version2.0に変更され，1980年に若干の訂正があって現在のVersion2.2となり，ＸＥＲＯＸ，ＨＰ，ＣＤＣなどメインフレーム・メーカーも続々と採用する今日に至っています．

CP/M

||

Control Program for Microcomputers

★ ＰＬ／Ｍ――現在でも最強力なインテル社のコンパイラの１つ．
ＣＰ／ＭもＰＬ／Ｍで書かれた部分が多くある．

CP/Mはほとんどの
マイクロコンピュータで走ります

CP／M はパーソナル・コンピュータを始め，ほとんどのディスク付きマイクロ・コンピュータ・システムで走ります．特にソフトウェア開発用のシステムは，8080又は Z80系であれば各社ともまず例外なく CP／M を採用し，最初から CP／M マシンとして発売しています．

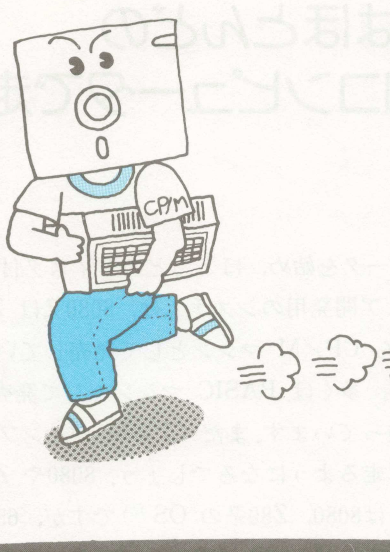
パーソナル・コンピュータの場合，多くは BASIC マシンとして発売されていますが，各社の代表的機種は，ほとんどで CP／M が走っています．まだ CP／M がインプリメント★されていない NEW モデルがあっても，近いうちに必ず走るようになるでしょう．8080や Z80系の CPU を使っているコンピュータは当然のこと（CP／M は8080，Z80系の OS★）ですが，6502や6809などの異種 CPU を持つコンピュータでも，CP／M を走らせることは可能なのです．たとえ異種 CPU であっても，コンピュータ・メーカーはなんとかして CP／M を走らせなければならない時代です．（6502の Apple II はすでに走っており，6809 の富士通 FM-8 でもCP／M が走ることが発表されている．）8080，Z80系以外のコンピュータが，〝Z80カード〟をオプションとして用意している意味の第1は，このCP／M を走らせるためなのです．

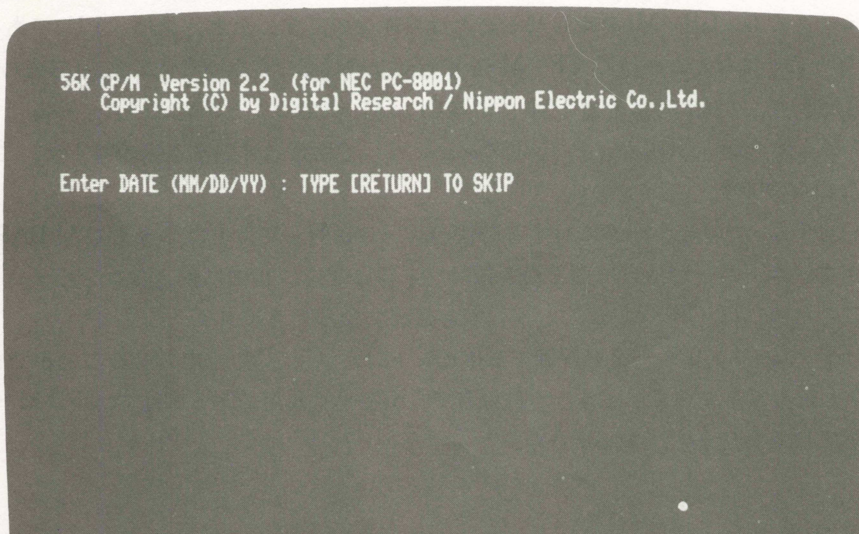
今日，パーソナル・コンピュータに対する時代のニーズに対して，もはや今までの BASIC マシンでは対応できず，ユーザーが独自に選んだ各種ソフトウェアを，自由に使えるコンピュータが望まれています．このためには CP／M が，どうしても必要になってきます．

今後のパーソナル・コンピュータを評価するポイントは，「CP／M を走らせることができるか．」が第1である，と言ってもよいでしょう．　付録Bで，現在すぐ利用できる国産の CP／M マシンを，リスト・アップしてあります．参照下さい．

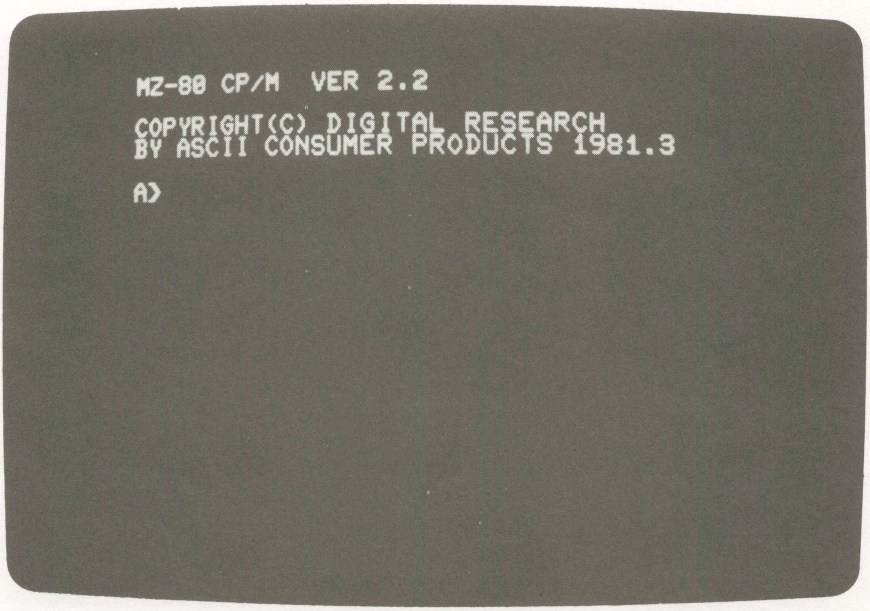
★ インプリメント――移植，実行すること．

★ OS――Operating System．1.3章参照．

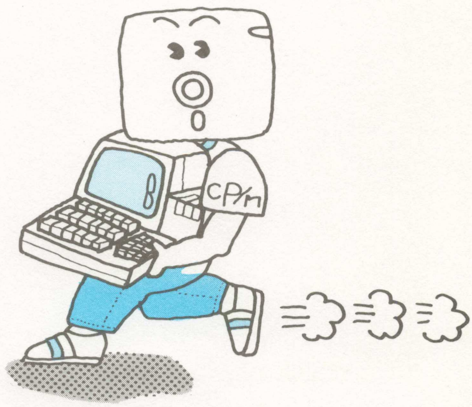




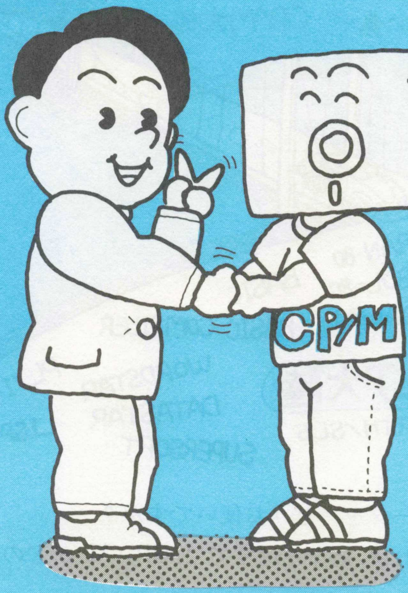
PC-8001 CP／M オープニング・メッセージ



MZ-80　CP／Mオープニング・メッセージ



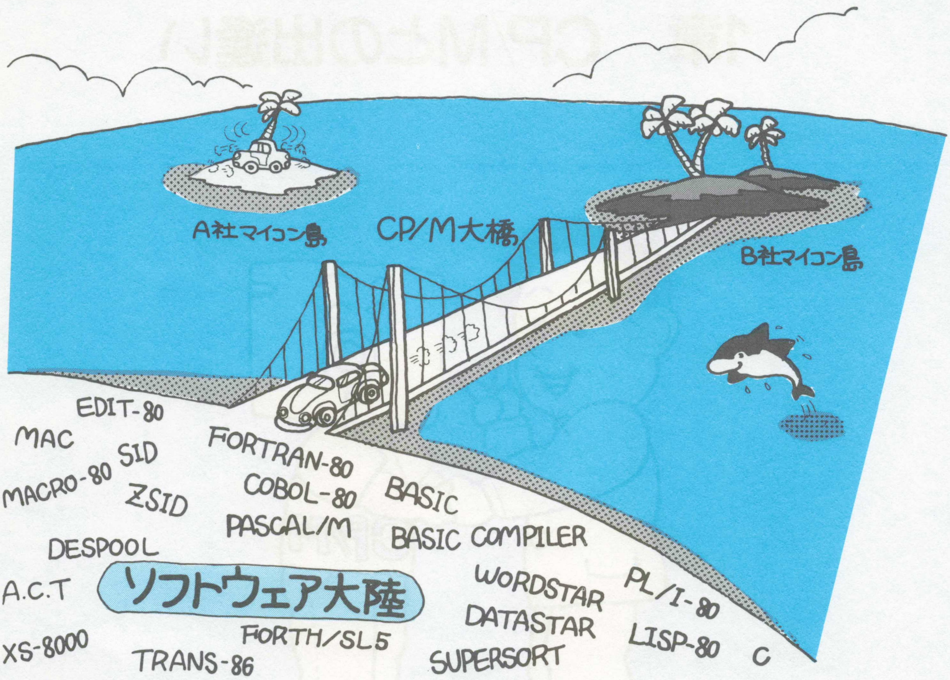
1章　CP/Mとの出逢い



CP/Mって何？
CP/Mはソフトウェア
CP/MはOS
CP/Mはソフトウェアのスタンダード・バス
CP/Mはマシン語開発ツール

1.1　CP/Mって何？

この一冊が，あなたと CP／M との素晴しい出逢いとなることを願って……



あなたはパーソナル・コンピュータを，どうお使いですか？

このイラストの意味が理解できれば，あなたはすでに，CP／M がどのようなものであるかの概念を，把握していると言えます．

残念ながら，多くのパーソナル・コンピュータは，そのままでは付属の BASIC 言語のみしか使うことができず，マイクロ・コンピュータ用に開発され，市販されている膨大な種類のソフトウェア群から，完全に孤立しています．しかし，あなたのパーソナル・コンピュータの能力は，BASIC 言語という単なる一方面のみに限られているわけではありません．本来は，もっともっとすごいことができるのです．

孤立しているあなたのパーソナル・コンピュータと，それを取り巻く膨大な種類のソフトウェア群．

CP／M は，この両者をインターフェースし，あなたのコンピュータを，膨大なソフトウェア群に取り巻かれた汎用スーパー・パーソナル・コンピュータに変えてしまうマイクロ・コンピュータの基本ソフトウェアなのです．

1.2　CP/Mはソフトウェア

CP／M はソフトウェアです．何らかの機械を含んでいるハードウェアではありません．例えば，NEC のパーソナル・コンピュータ，PC-8000ディスク・システムを例にとってみましょう．Fig-1.2.1をご覧下さい．

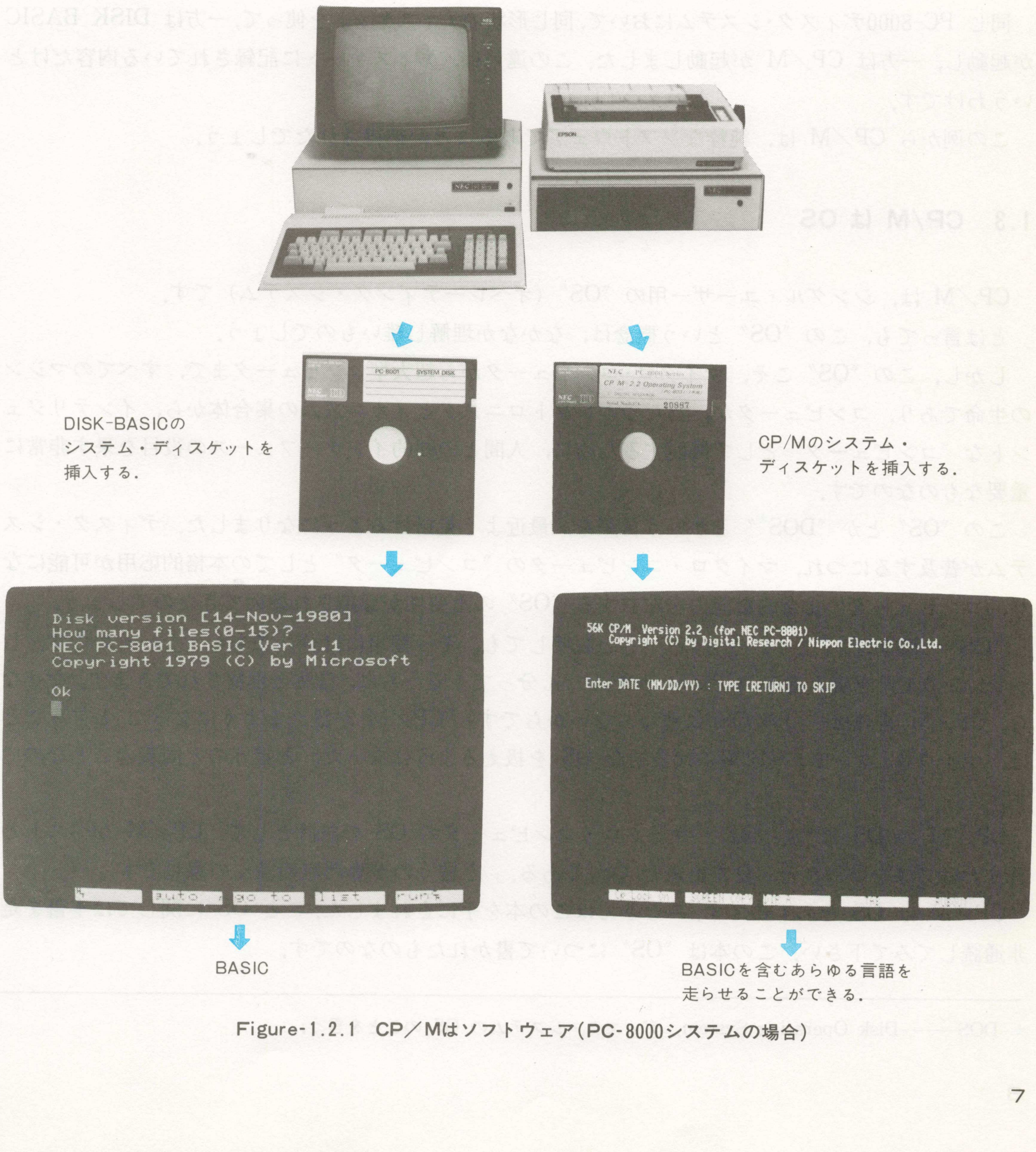


Figure-1.2.1　CP／Mはソフトウェア(PC-8000システムの場合)

PC-8000 DISK BASIC のディスケットと，PC-8000 CP／M のディスケットがあります．どちらもディスケット自体はまったく同じで，記録されている内容が異なるだけです．この２枚のディスケットを，両者同一の方法で起動させてみましょう．

どちらも起動は極めて簡単です．ディスケットをドライブ１に挿入して，リセット・ボタンを押すだけです．DISK BASIC と，CP／M がそれぞれのオープニング・メッセージの表示と共に起動しました．

同じ PC-8000ディスク・システムにおいて，同じ形状のディスケットを使って，一方は DISK BASIC が起動し，一方は CP／M が起動しました．この違いは，ディスケットに記録されている内容だけというわけです．

この例から CP／M は，純粋なソフトウェアであることが納得されたでしょう．

1.3 CP/M は OS

CP／M は，シングル・ユーザー用の〝OS〟（オペレーティング・システム）です．

とは言っても，この〝OS〟という概念は，なかなか理解し難いものでしょう．

しかし，この〝OS〟こそ，マイクロ・コンピュータから超大型コンピュータまで，すべてのマシンの生命であり，コンピュータが，ただのエレクトロニスクとメカニズムの集合体から，インテリジェントな〝コンピュータ〟として機能するために，人間との知的インターフェースの役目を果す非常に重要なものなのです．

この〝OS〟とか〝DOS★〟とかいう言葉を，最近よく見掛けるようになりました．ディスク・システムが普及するにつれ，マイクロ・コンピュータの〝コンピュータ〟としての本格的応用が可能になり，コンピュータの総合的な能力を左右する〝OS〟の重要性が認識され始めてきたのでしょう．

〝OS〟の概念は，「こういうものです」と説明しても，そう簡単には分ってもらえないものです．しかし，この１冊を読み進み，CP／M がだんだん分ってくるうちに，自然と理解されてきます．なぜなら，CP／M 自身は一つの OS に他ならないからです．「CP／M が扱えるようになった」と言うことは，マイクロ・コンピュータ用の代表的な OS を扱えるようになった，と言うのと同義のことなのです．

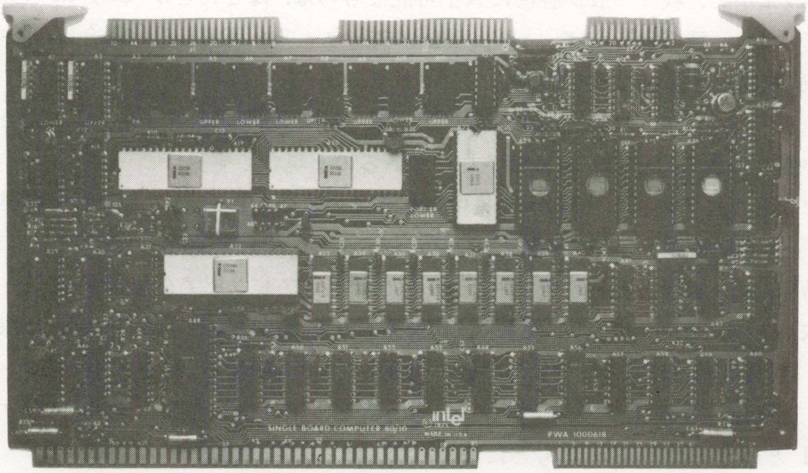
CP／M の OS については，「マイクロ・コンピュータの OS の設計として，CP／M がベストと言うわけではないけれど，良く出来た OS である．」と言うのが専門家の多くの意見です．

CP／M は OS そのものです．みなさんはこの本を手にされました．こと OS に関しては本書を是非通読してみて下さい．この本は〝OS〟について書かれたものなのです．

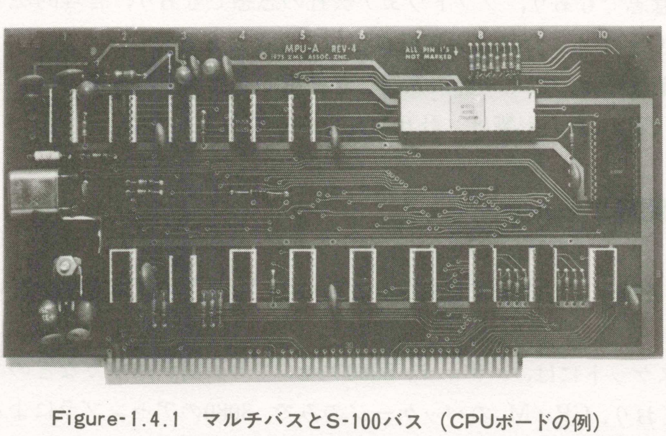
★ DOS———Disk Operating System．ディスク・システムの OS のことを言う．

1.4 CP/Mはソフトウェアのスタンダード・バス

〝バス〟と言えばハードウェアにおいては，Fig-1.4.1に示すようなインテル社のマルチ・バスとか，IEEE のS-100 バスなどが有名であり，それらのバス★上で働く各種ボード類が，多くの会社から発売されています．よって，コンピュータ・システムにこれらのバスを採用すると，システムを設計したり拡張したりする場合，任意のボードが容易に入手でき，自由なシステム作りが可能となるため，汎用性の面からも大変有利になります．



マルチバス



S-100バス

Figure-1.4.1 マルチバスとS-100バス（CPUボードの例）

★ バス――コネクタに集められているアドレス，データ，コントロール，その他の信号線路群．

このようにハードウェアの有力な〝バス〞上で働く，多くの種類のボードが，各社から発売され，それをユーザーが目的に合わせて自由に使用する，――このことが，そっくりソフトウェアにも当てはまるのです．

CP／M という有力なソフトウェア上の〝バス〞（バス＝OSと言い替えられる）に合わせて，多くのソフトウェア会社が，多種多様のソフトウェアを発売しています．よってユーザーは，自分のコンピュータ・システムの OS に CP／M を採用することにより，容易に任意のソフトウェアを走らせることが可能になります．このことを指して「CP／M はソフトウェアのスタンダード・バス」と表現しているわけなのです．

今日，各種ソフトウェアで，我々一般が容易に入手可能なものは，ほとんどが CP／M 上で実行されるように作られています．言い替えれば，いろいろなソフトウェアを利用したい場合は，まず CP／M を走らせる必要がある，と言うことです．

マイクロ・コンピュータのソフトウェアの中心は CP／M です．CP／M を中心として各分野，各種のソフトウェアが集り，さらにどんどん蓄積されつつあるのが現状です．

なぜこのようにソフトウェアの CP／M への集中が起きているのでしょう．これは簡単な理由です．マイクロ・コンピュータの成長期に，一つの優れた OS が生まれ，それが普及し始める．ソフトウェア会社は，広い市場を求め，普及し始めたその OS の下で走るソフトウェアを発売する．⟶それらのソフトウェアを求めるユーザーが，そのソフトウェアを走らせるために CP／M を OS に採用する．⟶ますます CP／M 利用者は増加する．CP／M に付属している開発ツールを利用するユーザーも同様に増えていく．このような現象がループになり，もう8ビット・コンピュータのソフトウェアはほとんどが CP／M という一つの OS のもとに集まってしまったのです．

この現象は，ユーザーの意志でもあり，ソフトウェア会社の思惑でもあり，合理的かつ経済的であり，CP／M が良く出来た OS であるだけに大変好ましいことと言えるでしょう．

そして，結果として数年前までは，ごく一部の人達しか使用できなかった各種のソフトウェアを広く安価に一般に開放し得た CP／M の影響は，計り知れないものがあります．

1.5 CP/Mはマシン語開発ツール

CP／M は OS であり，各種ソフトウェアを走らせるための〝バス〞の役目を果していることは，前節で述べました．

が一方，CP／M のディスケットには，エディタ，8080アセンブラ，同デバッガなどのユーティリティー・プログラムが含まれており，CP／M のパッケージのみで，8080のアセンブラによるソフトウェア開発ができます．

エディタ（CP／M のエディタは〝ED〞と言うプログラム名）は，スクリーン・エディタではなく，

ポインタ形式のものですが，十分に強力で，その使い易さは大型マシンのエディタと比較しても引けを取りません．このエディタで，文章やプログラムの作成が出来ます．今まで本格的なエディタを使う機会がなかった方は，この強力な多くの機能に目を見張ることでしょう．

CP／M のエディタの内部コマンドは20以上もあり，最初は覚えるのに苦労します．しかし，なかなか整ったコマンド体系を取っており，覚えてしまうと大変使いよく，もっと強力な別のエディタ・プログラムを持っている人でも，この CP／M のエディタを常用している人が多いのです．

アセンブラ（CP／M のアセンブラは〝ASM〟と言うプログラム名）は，マクロ機能はありませんが，ラベルは16文字まで識別可能で，〝IF〟，〝END　IF〟による条件付きアセンブルができるなどの，いくつかの拡張機能があり，インテル HEX 形式のオブジェクト・コードと，アセンブル・リストを出力する使い易いアセンブラです．マクロ機能が必要なユーザーや Z80をアセンブルするには，別売の〝MAC★〟や〝MACRO-80★〟などを使用します．

CP／M のデバッガは〝DDT〟（ダイナミック・デバッギング・ツール）と呼ばれており，ダンプ，メモリ内容の変更や，フィル(任意のメモリを同一コードで埋める)，ブロック移動などの通常のモニタ機能の他，ダイレクト・アセンブラ，逆アセンブラや，テスト・プログラムのトレース実行と１ステップ毎の各レジスタの値の表示，ブレーク・ポイントを設定しての実行などの機能を備え，デバッグ作業がダイナミックに行える強力なものです．

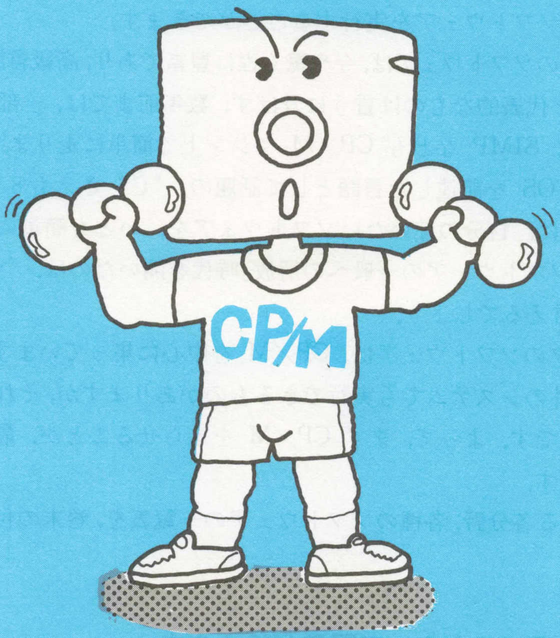
その他には，〝LOAD〟と呼ぶ，インテル HEX 形式のオブジェクト・ファイルを，実行可能な純マシン・コードに変換するプログラムや，〝PIP〟と呼ぶ，周辺装置のデータ転送プログラムなどが含まれており，CP／M のパッケージだけで，他のソフトウェアを購入することなく，8080のアセンブラによるソフトウェア開発を能率よく行うことが可能です．もちろんアッパー・コンパチである Z80上でも実行できることは言うまでもありません．

マシン語開発に関して，もう一つ重要な機能があります．それはシステム・コール，またはファンクション・コールと呼ばれるもので，CP／M の OS のいくつかの機能を，ユーザーがサブルーチンコールにより自由に利用できる非常に有用で強力な CP／M のシステム・サービスのことです．

このシステム・コールについては〝応用CP／M〟で取り上げますが，CP／M 上で走るマシン語を開発する上で必要不可欠の大変便利な機能であるとだけ記憶しておいて下さい．

★ MAC，MACRO-80———付録Ａ参照．

2章　CP/Mで何が出来るか?



ほとんどすべてのソフトウェアが実行可能になります
各種高級言語でのソフトウェア開発が行えます
16ビットを含む各種CPUのマシン語開発が行えます

この章の内容は，第１章の「CP／M って何？」で述べられていることを，反対側から見れば，それが答になりますが，あえて，〝何が出来るか〟の面から解説してみましょう.

2.1 ほとんどすべてのソフトウェアが実行可能になります

第１章の1.4で述べたように，CP／M はマイクロ・コンピュータの標準 OS であり，各種ソフトウェアを走らせるためのスタンダード・ソフトウェア・バスでもあります．よって，まず CP／M を走らせることにより，あなたのコンピュータで，各種の高級言語を始め，ソフトウェア開発ツール，ユーティリティー・ソフト，ビジネス・ソフト，アプリケーション・ソフトなど，一般に知られているほとんどすべての種類のソフトウェアを実行することができます.

マイクロ・コンピュータのソフトウェアは，今や驚く程に豊富であり，高級言語を例に取ると，COBOL, FORTRAN, PL／I など代表的なものは言うに及ばず，数年前までは，一部の研究所や大学でしかお目に掛れなかった LISP, SIMP なども CP／M マシン上で簡単に走ります．最近は〝UNIX〟と言う16ビット・マシン用の OS を記述した言語として話題の，〝C〟さえも８ビットの CP／M マシンで走るのです．このように，自分の使いたいソフトウェアを，いとも簡単に自分のコンピュータで走らせることができる，「ソフトウェアの一般への解放」時代を開いたのは，〝マイクロ・コンピュータ＋CP／M〟によるものと言えるでしょう.

マイクロ・コンピュータのソフトウェアは，CP／M を中心に集っています．その中の代表的なものの幾つかは，CP／M 以外のシステムでも実行できるものがありますが，それらのソフトウェアの種類は非常に少ないのが現状です．よって，まず CP／M を走らせることが，最良の環境を経済的に作り出す唯一の手段と言えます.

CP／M 上で実行される各分野，各種のソフトウェアの一覧表を，巻末の付録Ａにまとめてありますので参照下さい.

2.2 各種高級言語でのソフトウェア開発が行えます

巻末付録Ａの「CP／M で走るソフトウェア一覧表」にもあるように，我々がよく耳にする言語は，ほとんど CP／M 上で実行できます．そして，それらの品質は，例えば〝FORTRAN-80〟は ANSI-66★に準処，〝CIS-COBOL〟は ANSI-74★に準処，という具合にプロフェショナル・ユース用のものです.

数年前までは，ミニコンピュータの中でもソフトウェアの充実したもの（ミニコンピュータは，い

★ ANSI-66，ANSI-74———アメリカ国立標準協会．4.2章参照.

くつかの著名な機種を除くと，ソフトウェアの〝充実〞に関しては，実に貧弱なものが多い）や，大型コンピュータでしか使えなかった言語を使ってのソフトウェア開発が，パーソナル・コンピュータ上で，どんどん行われるようになりました．

最近では，最終的には大型コンピュータで実行する高級言語のプログラムを，CP／M マシンで開発するケースも増えています．時間の掛るデバッグ作業などの開発に関しては，ランニング・コストが只同然（大型マシンと比べて）のCP／M マシンで行い，バグのない〝クリーン・ソース，クリーン・データ〞にしてから，大型コンピュータにロードする，という開発法です．近い将来，パーソナル・コンピュータのデータ通信によるネットワークが発達すると，このようなことも日常化するのではないかと思われます．

また逆に，今までミニ・コンピュータや，大型コンピュータによって開発され，蓄積されている各種言語によるプログラムを，大巾な書き替えを必要とせず，そのままマイクロ・コンピュータ上で働かせることができます．今後ますます上位マシンにより開発・蓄積された優れたソフトウェアが，マイクロ・コンピュータ用にダウン・ロード★され，我々の CP／M マシンで実行されるようになるでしょう．

2.3　16ビットを含む各種 CPU のマシン語開発が行えます

CP／M は，そのパッケージに，エディタ，アセンブラ，デバッガなどの8080アセンブラによるソフトウェア開発ツールを含んでいることは，第1章の1.5で述べました．CP／M のアセンブラは，マクロ機能がなく，かつリロケータブルなオブジェクト・コードを生成することができませんが，これらの機能を必要としないユーザーは，CP／M に含まれている開発ツールのみで，十分能率的な作業を行うことができます．

さらに高度な機能を必要とするユーザーには，巻末付録Aのソフトウェア一覧表の〝開発ツール〞欄に示されている各種ソフトウェアを使用することができます．

特にマイクロソフト社の8080／Z80用アセンブラの〝MACRO-80〞は，リロケータブル・オブジェクト・コードを生成でき，このコードがリロケータブル・オブジェクト・コードのインダストリアル・スタンダードとして，広く用いられています．また，各種コンパイラから出力されるオブジェクト・コードも，これとコンパチビリティを持たせたものが主流となっており，それぞれを自由に結合することが可能です．要するに，COBOL や， FORTRAN や BASIC コンパイラ，それに，MACRO-80アセンブラなどで得られたオブジェクトを，任意に組合わせて一本のプログラムとすることが，容易に行えるわけです．

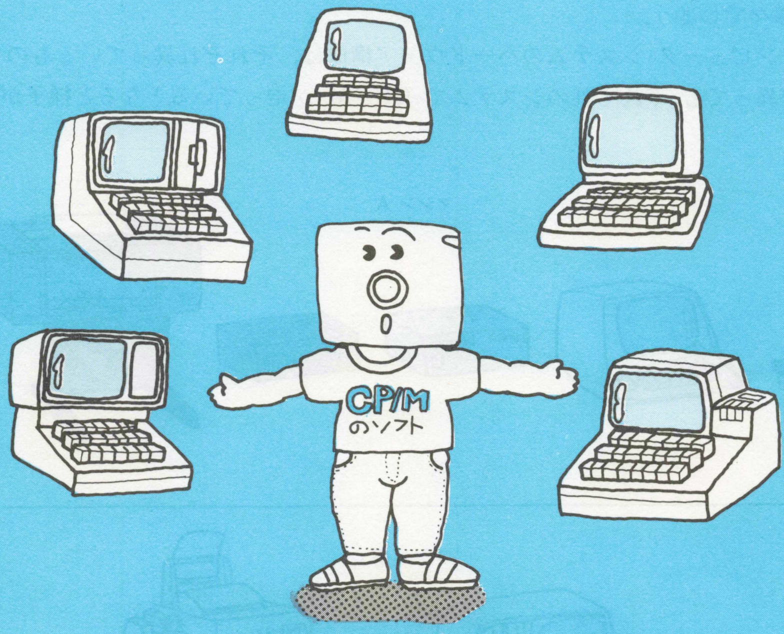
★ ダウン・ロード――メディア変換すること．

この機能は非常に重要なもので，例えば，メイン・モジュールは COBOL で書かれたオブジェクトで，サブ・モジュールは BASIC コンパイラによるオブジェクト，という組合せが簡単に出来るのです.

他の開発ツールには，〝SID〞，〝ZSID〞と呼ぶ強力な8080又は Z80用のシンボリック・インストラクション・デバッガなども用意されており，CP／M 付属の〝DDT〞より一段と強力なデバッグが行えます.

さらに，ソフトウェア一覧表の〝クロス・ソフト〞の欄に示されている各種クロス・アセンブラや，トランスレーターがあり，CP／M マシン上で，16ビットを含む各種 CPU のアセンブルを行ったり，8ビット用のソース・プログラムで，16ビット CPU のオブジェクト・コードを自動的に生成したりすることができます.

3章　CP/M上のソフトウェアは全CP/Mマシンで共通です。



ハードウェアの異なるCP/Mマシンとソフトウェア
種類の異なるディスケットとソフトウェア

CP/M 上のソフトウェアは，よほど特種なプログラムでない限り，ハードウェア構成の異る各社各様の CP/M マシンで，そのプログラムに何の変更も加えずに，実行することができます．この特徴が CP/M の最大のメリットでもあるわけです．本章では，異る CP/M マシンと応用するソフトウェアの関係，ディスケットの違いによるソフトウェアの互換性の問題などを解説します．

3.1 ハードウェアの異なるCP/Mマシンとソフトウェア

CP/M が走るマイクロ・コンピュータ・システムのハードウェア構成は各社各様です．例えば，フロッピー・ディスクについては，標準 8 インチ・サイズ，5¼インチのミニサイズ，おまけに両面だの倍密度だの様々ですし 5 メガ・10メガのハード・ディスクを付けたものまであります．CPU 周辺の I/O 構成なども様々でしょう．

このようにコンピュータ・システムのハードウェア構成は，それぞれ異っているものです．しかし，ハードウェアが異っても，それぞれのシステムで CP/M が走っているとなると様子が変ってきます．

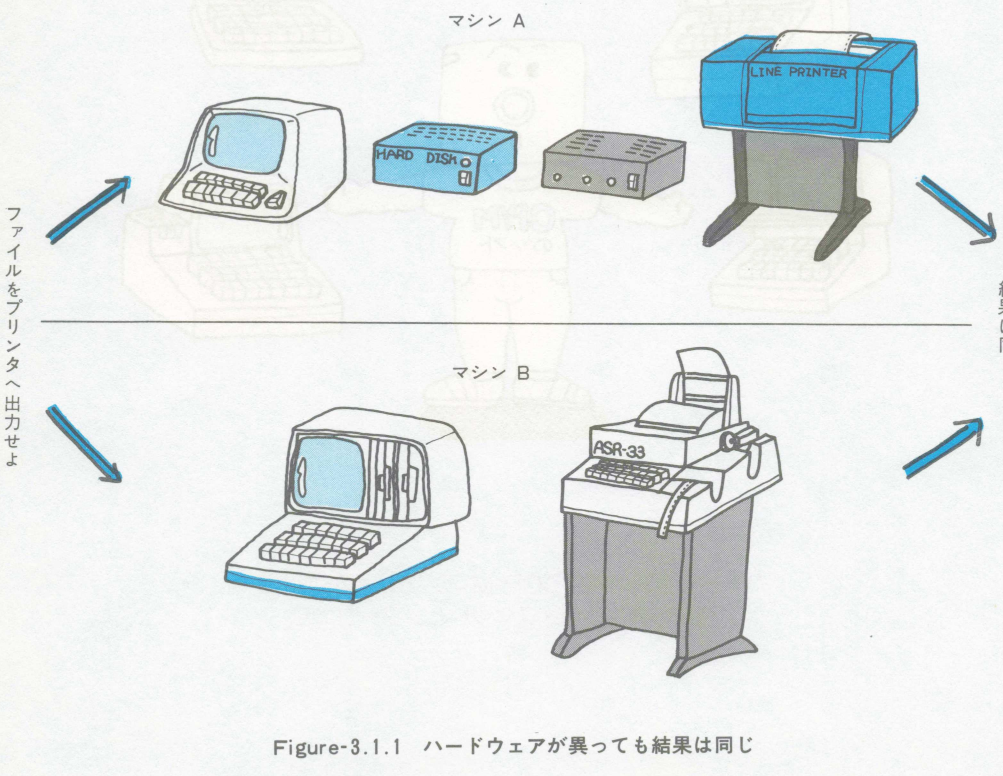


Figure-3.1.1 ハードウェアが異っても結果は同じ

CP／M が走っているシステムは，CP／M 上のソフトウェアから見ると，それぞれは異ったハードウェアシステムであっても，〝CP／M マシン〟という同一のロジカル的（論理的）なハードウェアを持つシステムとみなされるのです．

一例を挙げると，CP／M 上で実行されているプログラムの一部に，「ディスク上の，あるファイルをプリンタへ出力せよ」という指示があったとします．

この指示が書かれている同一のプログラムを，CP／M マシンAと，Bに与えたとします．マシンAの周辺装置がハード・ディスクであることや，高速のライン・プリンタであること，そしてマシンBが時代物の ASR-33テレタイプを優雅に使っていることなどは，プログラムは全く知らないことで，そのことの指示はどこにも書かれていません．

結果はどうでしょう．

ハードウェアが異っても，同じプログラムでそれぞれのプリント用紙に，目的のファイルが間違いなくプリント・アウトされます．Fig-3.1.1がその様子を表わしたものです．

この例のように，CP／M 上のソフトウェアは，コンピュータ・システムのハードウェアが異っていても，共通に使用できるのです．

これはCP／M の基本設計が，8080や Z80系のどのようなコンピュータ・システムであっても，容易に適合できるようにと最初から狙って作られた OS であるからです．この点が特定のマシン専用のスタンド・アローンと呼ばれる OS と異るところであり，CP／M が今日の隆盛をみた要因でもあるのです．

このことをもう少し詳しく説明しましょう．CP／M 本体の構成は Fig-3.1.2に示すように，それぞれのコンピュータ・システムのハードウェアに関係する部分と，関係しない部分とに分けることができます．

関係する部分は，〝BIOS〟（Basic I／O System，「バイオス」と呼ぶ人が多い）と言い，各システム個有のキーボードや CRT ディスプレイ，ディスク・ドライブのコントロール，プリンタや他の I／O ポートのコントロールなどのルーチンを収めてあります．通常は，キーボードやディスプレイ，ディスク・ドライブなどのハードウェアは，メーカーが異ればコントロール・ソフトウェアも異ります．よって，この〝BIOS〟だけは，それぞれのコンピュータ・システムに合わせて，独自に作り上げなければならない部分です．

CP／M のその他の部分はハードウェアに関係せず，全マシン同一です．言い替えれば，〝BIOS〟部分さえ目的のコンピュータ・システムに適合させれば，どのようなシステムであっても CP／M を走らせることができるわけです．

CP／M 上の一つのプログラムが，異る CP／M マシン上で変更なしに走るわけもこの〝BIOS〟にあり，共通のプログラムであっても，〝BIOS〟が，それぞれのハードウェアに合わせた変換を行っているからなのです．

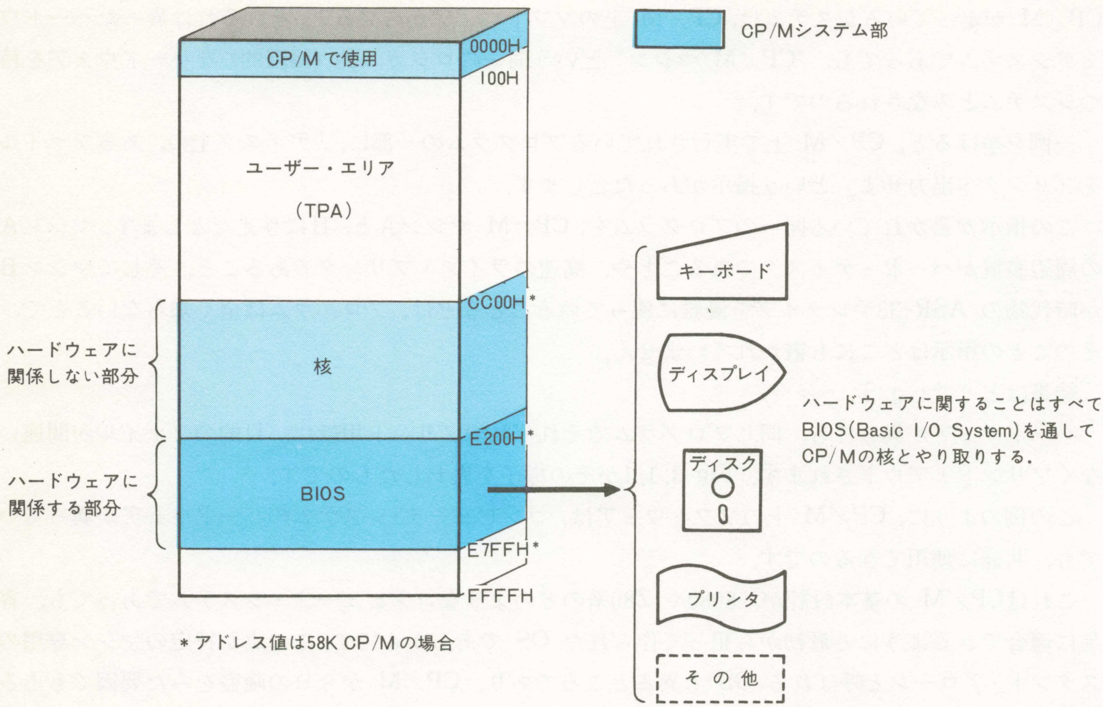


Figure-3.1.2 BIOSの役割

3.2 種類の異なるディスケットとソフトウェア

一つの例を挙げましょう．CP／M 上のソフトウェアで，ポピュラーなものの一つに，マイクロソフト社の〝BASIC COMPILER〞という，MBASIC★ とコンパチブルのソース・プログラムをコンパイルできるコンパイラ★があります．これらのソフトウェアのすべての種類は，マイクロソフト社からも，日本の代理店からも，基本的には，８インチ，片面単密度の標準フロッピー・ディスケット（このフォーマットが IBM 3740と呼ばれる）で供給されています．

よって，ユーザーが使用しているコンピュータ・システムのディスク・ドライブが，８インチの標準フロッピー用であれば，どのような CP／M マシンでも何も考えることなく，そのままで実行することができます．しかし，ディスク・ドライブの形体が異れば，８インチ標準フロッピー・ディスケットそのままの形では，使用することはできません．

★ MBASIC，コンパイラ――12章で実行例を示してあります．

このような場合は，〝メディア変換〟を行います．メディア変換のことを〝ダウン・ロード〟とも言い，あるシステムのディスク上の（プログラム）ファイルを，異った種類のディスク・ドライブを持ったシステムのディスク上に〝落し込む〟（転送する）ことを言います．

このメディア変換の様子は，オーディオ・テープのダビング（コピー）に例えてもよいでしょう．ある曲がオープン・リールの4トラック・テープに録音されていたとします．これをカセット・テレコで楽しみたい人は，カセットにダビングをすればよいし，38センチ2トラックのオープン・リールで楽しみたい人は，それにダビングをしておけば，いつでも自分の装置で聴くことができます．（いつでもそのソフトウェアを自分の CP／M マシンで実行することができます）．

このようにメディア変換の作業は，転送しようとするファイルにはいっさい手を加えず記録媒体だけが変って中身は同じ，と言うことを行うわけです．そして，ディスクの形状が変っただけで，中身が同じソフトウェアは，それぞれの CP／M マシンで実行することができるのです．(Fig-3.2.1参照)

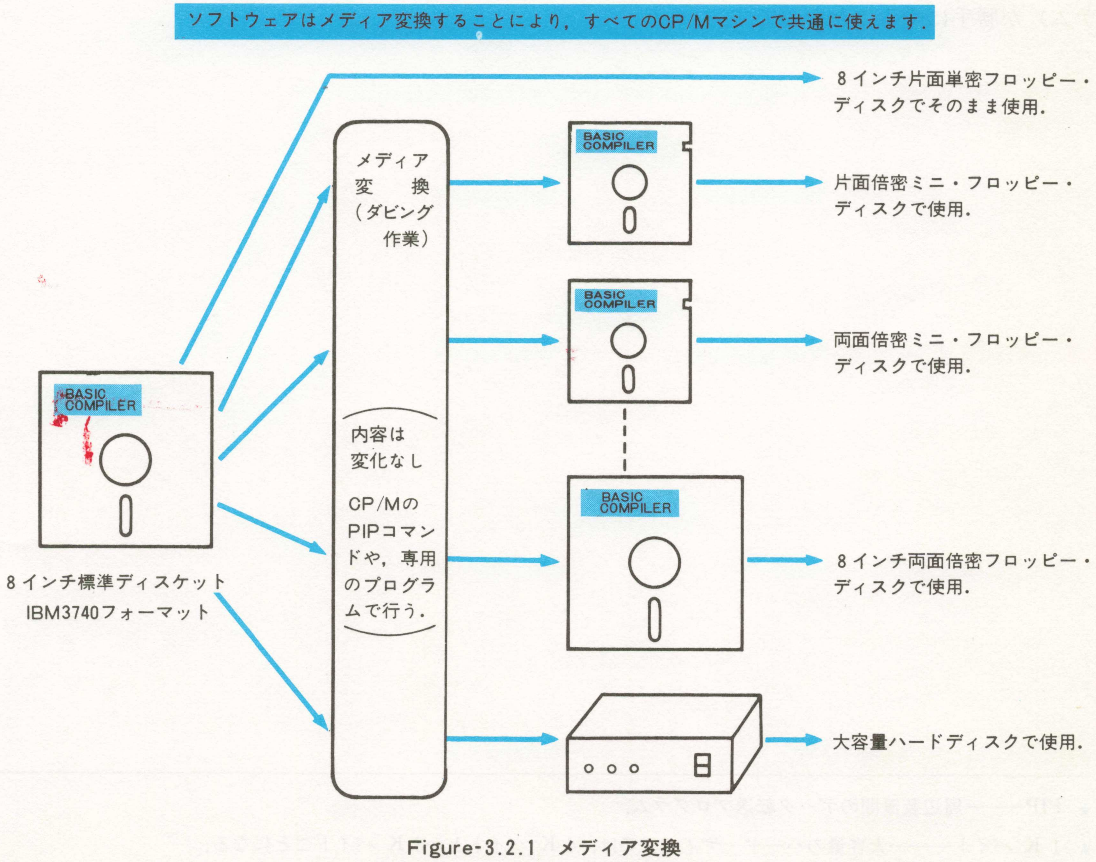


Figure-3.2.1 メディア変換

実際には，各自がメディア変換を行う必要はなく，各社のパーソナル・コンピュータの CP／M 用に，すでにメディア変換されたソフトウェアが豊富に用意されていますので，ユーザーは，自分のマシン用のソフトウェアを，即入手することができます．

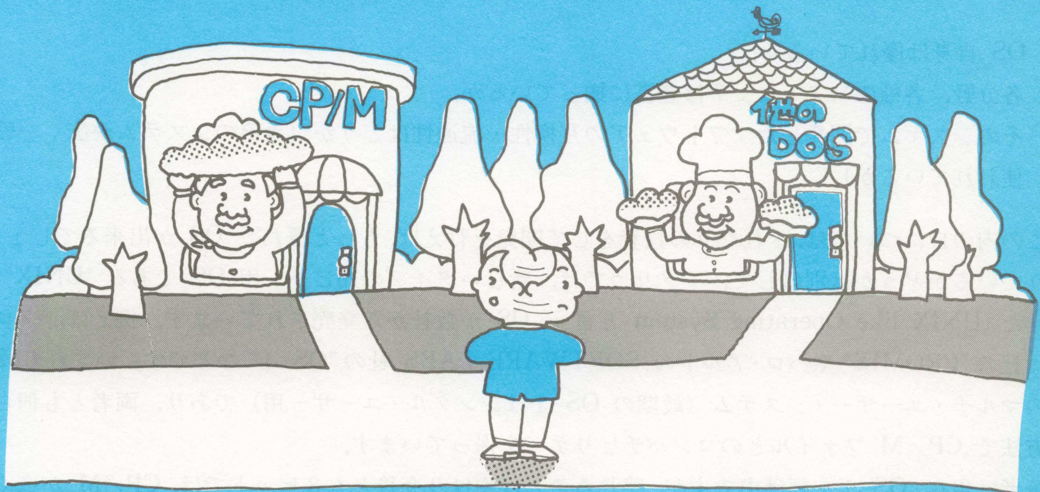
なおメディア変換の実際については，CP／M コマンドの〝PIP★〟や，メディア変換の専用プログラムなどで行いますが，詳しくは本シリーズの続巻で解説します．

注）ディスク上のファイルについて，うるさいことを言うと，ディスク・ファイルの管理は〝ダイナミック・アロケーション〟と言って，オーディオ・テープのように1曲（1つのファイル）1曲が，必ずしも連続して収まっているのではなく，曲の小節（？）ごと（1Kバイト★ごと）にディスケット上にばらばらに散在していてもよいのです．しかしユーザーは，目的のファイルが，ディスケット上のどこに散在しているのだろう，とか，どうやって拾い集めて連続させればよいか，などと考える必要はありません．このようなことはすべて CP／M の〝DOS〟（ディスク・オペレーティング・システム）が勝手にやることなのです．

★ PIP———周辺装置間のデータ転送プログラム．

★ 1K バイト———大容量のハード・ディスクでは，4K バイトとか8K バイトごとになる．

4章　CP/Mと他のシステムとの比較



なぜCP/Mを選ぶのか？
CP/M上で走るソフトウェアの品質

4.1 なぜCP/Mを選ぶのか

一般のユーザーが入手可能な範囲では現在，CP／M以外のシステムで，CP／Mと対等に比較できるようなシステムは残念ながら見当りません．6800系でも専用の OS がありますが，普及とソフトウェアの充実など，まだまだ今後に期待しなければなりません．ほかには，2～3の言語が特定のパーソナル・コンピュータで実行可能なシステムもありますが，これらは，スタンド・アローン・システム（自分自身だけで実行可能なシステム）と呼ばれ，逆に言えば，そのマシン専用に作られたソフトウェアしか実行できないシステムであり，ソフトウェアの種類も少ないのが現状です．もちろん，それだけが使えればよいユーザーには，それでもよいでしょう．

CP／M や，他のシステムを論じる場合，次の3点について総合的に評価しなければなりません．

1) OS 自身は優れているか．
2) 各分野，各種のソフトウェアは豊富に揃っているか．
3) そのシステムで開発したソフトウェアの互換性・流通性はどうか（＝そのシステムが広く一般に使われているか）．

この内の1)については，それなりの投資をして開発を行えば，もっと優れた OS が出来るでしょう．優れているかどうかは別にして，アメリカでは，新しいタイプの16ビット用 OS である〝UNIX〟に似せた（UNIX like Operating System と言う）OS が数社から発売されています．例えば，クロメムコ社の〝CROMIX〟やパロ・アルトの SOFTWARE LABS 社の〝OS-1〟などです．いずれも Z80用のマルチ・ユーザー・システム（最期の OS-1はシングル・ユーザー用）であり，両者とも何らかの方法で CP／M ファイルとのコンパチビリティを保っています．

筆者は現在，OS-1を評価中ですが，感じることはやはり今後とも8ビットでは，CP／M が広く使われていくことに変りはないだろう，ということです．なぜなら8ビットのマイクロ・コンピュータにおいては，OS 自体を，CP／M や MP／M★以上に高度なものにして，そのコマンドや，ファイル構造を，一般に馴みにくいものにしてしまう必要性はないと思うからです．簡便でコンパクト，それでいて十分な機能こそ，8ビットのマイクロ・コンピュータ用 OS の前提であると思うのです．この議論はマイクロ・プロセッサが出回り始めた頃のチップ自体の優劣の議論と似ています．

しかし，このような，OS 自体の優劣の議論より，現実には，2)，3)の問題の方が，重要です．

結局2)，3)の点において CP／M より良い環境のシステムは現実には皆無であり，今後とも現われる可能性はほとんど無いでしょう．そして CP／M ユーザーの数は増加する一方であり，ソフトウェ

★ MP／M――マルチ・ユーザー・システムの CP／M．13章参照．

アも一層充実していくことは間違いないと思われます．

4.2 CP/M上で走るソフトウェアの品質

CP／M では，マイクロ・コンピュータ用の最高クラスの各種ソフトウェアが走っています．

例えば Micro Focus 社の CIS-COBOL を例にとってみましょう．COBOL の世界的な仕様規格は，CODASYL(コダシル) 委員会がまとめ，それを ANSI(アンシー)(アメリカ国立標準協会)が正式に決定します．COBOL には現在，ANSI-74という規格があり，Fig-4.2.1に示すように low から high まで4つのレベルがあります．

CIS-COBOL ver.4 は下から2番目のレベルの COBOL です．しかし1981年9月には ver.5 が発売され，これは何と最高レベルの COBOL なのです．この最高のレベル4は，大型コンピュータで走る COBOL と同等であり，通常のミニコンで走っている COBOL がレベル2程度であるので，正に CP／M 上で，最高の品質のソフトウェアが走る例と言えます．

この CIS-COBOL は付録Aのソフトウェア一覧表にもあるように，国内で容易に入手できます．

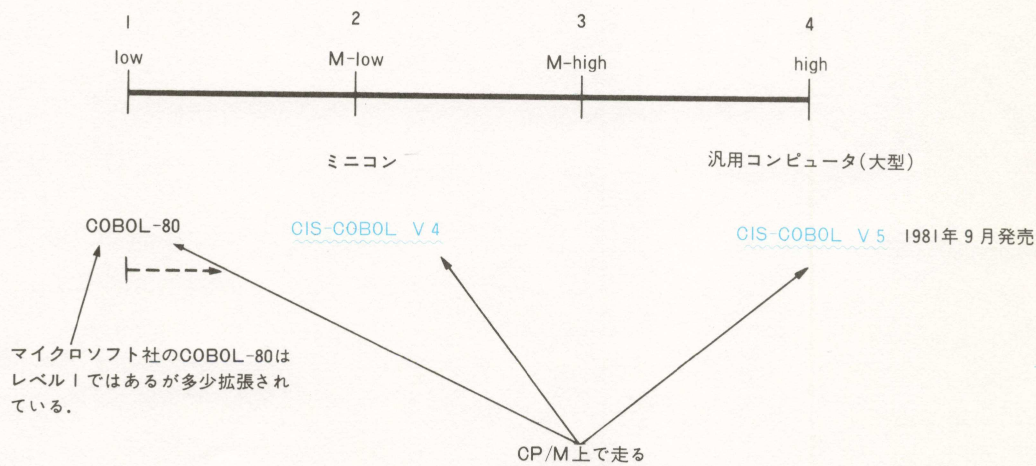
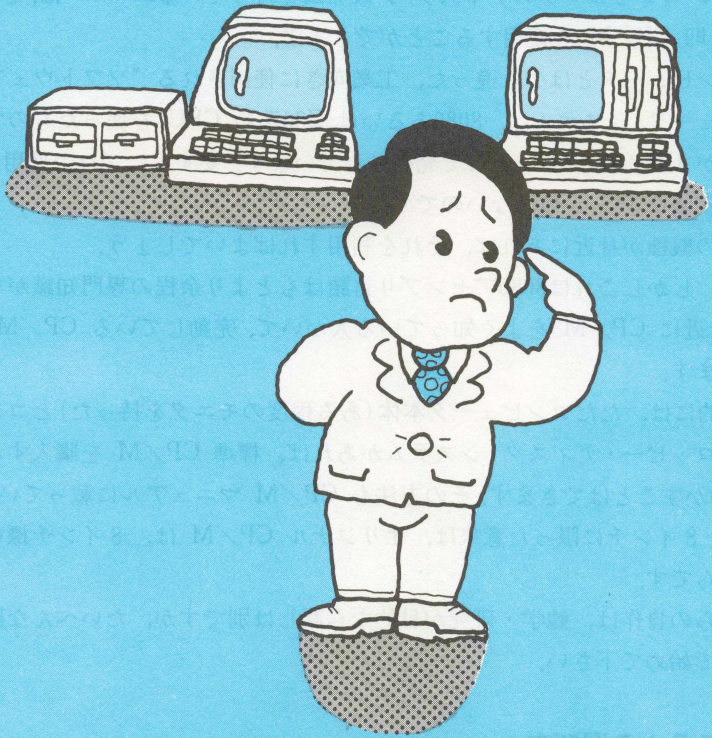


Figure-4.2.1 ANSI-74におけるCOBOLのレベル

5章　CP/Mを導入するにあたって



CP/Mを導入するには？
CP/Mシステムの選び方
CP/Mを買うと何が付いてくるのか？

5.1 CP/Mを導入するには？

まず一般的には，CP／M の走っているパーソナル・コンピュータを使うことが良いでしょう．大手各社も遅ればせながら，CP／M という，この国際的な汎用 OS の重要性に気付き，非常に力を入れてきました．パーソナル・コンピュータの各社の主要機種は，ほとんどで CP／M が走っていますので，目的に合った周辺装置や，付加機能を持ったものを選ぶことができます．

すでにパーソナル・コンピュータのディスク・システムを持っているユーザーは，その機種用の CP／M を購入すれば，即 CP／M を使用することができます．

パーソナル・コンピュータとは少し違った，工業向きに使用される〝ソフトウェア開発システム〟と言われるコンピュータ・システムは，8080あるいは Z80系の CPU を使ったものであれば，以前からほとんどの機種が CP／M マシンとして発売されています．要するにプロの使用に耐え，ソフトウェアが豊富でかつ安価な OS が他にないのです．

よって，これらの機種が身近にあれば，それを利用すればよいでしょう．

最後は自作です．しかしこれは8080アセンブリ言語はもとより余程の専門知識がないと，まず不可能です．最低限，身近に CP／M をよく知っている人がいて，完動している CP／M マシンを利用できることが望まれます．

CP／M は基本的には，ただコンピュータ本体（ある程度のモニタを持った）とコンソール，それと 8 インチ標準のフロッピー・ディスク・システムがあれば，標準 CP／M を購入することによりゼロから CP／M を動かすことはできます．その方法も CP／M マニュアルに載っています．ここでフロッピー・ディスクを 8 インチに限った意味は，オリジナル CP／M は，8 インチ標準ディスケットで供給されているからです．

しかし，ゼロからの自作は，勉学・研究が目的なら話しは別ですが，たいへんな困難を伴いますので，その心づもりで始めて下さい．

5.2 CP/Mシステムの選び方

パーソナル・コンピュータを本格的に応用し始めると，付属の BACIC 言語では能率が上がらなかったり，周辺装置のサポートなどいろいろな限界を感じたり，ほかのもっと適した言語や様々のアプリケーション・プログラムを使いたくなるなど，現状システムでの行きづまりが必ず現われてきます．そこで一般的に最も優れた環境を持つ CP／M システムを採用することになると思いますが，その場合の要点を述べておきましょう．

まず，CP／M には〝標準 CP／M〟というものがあります．何が標準かと言うと，本家である Digital

Research 社が直接発売しているもので，日本では極東総代理店であるマイクロソフトウェア・アソシエイツなどで〝標準 CP／M″として，8 インチの標準フロッピー・ディスケットで発売しているものです．PC-8000用とか if800用とか他にも各種ありますので，注文する時にはそれぞれ指定しなければなりません．

標準 CP／M は，インテル社の開発マシンである MDS-800用として，BIOS 部が書かれていますが，このディスケットがパーソナル・コンピュータ用の CP／M はもとより，すべての CP／M の基になっています．

現在，各社のシステムの専用として発売されている CP／M は，この〝標準 CP／M″を基にして，それぞれのマシン用に BIOS 部を変更し，独自に，それぞれのマシン特有の機能を拡張したり，ユーティリティー・プログラムを付属させたりして，特色を出しています．

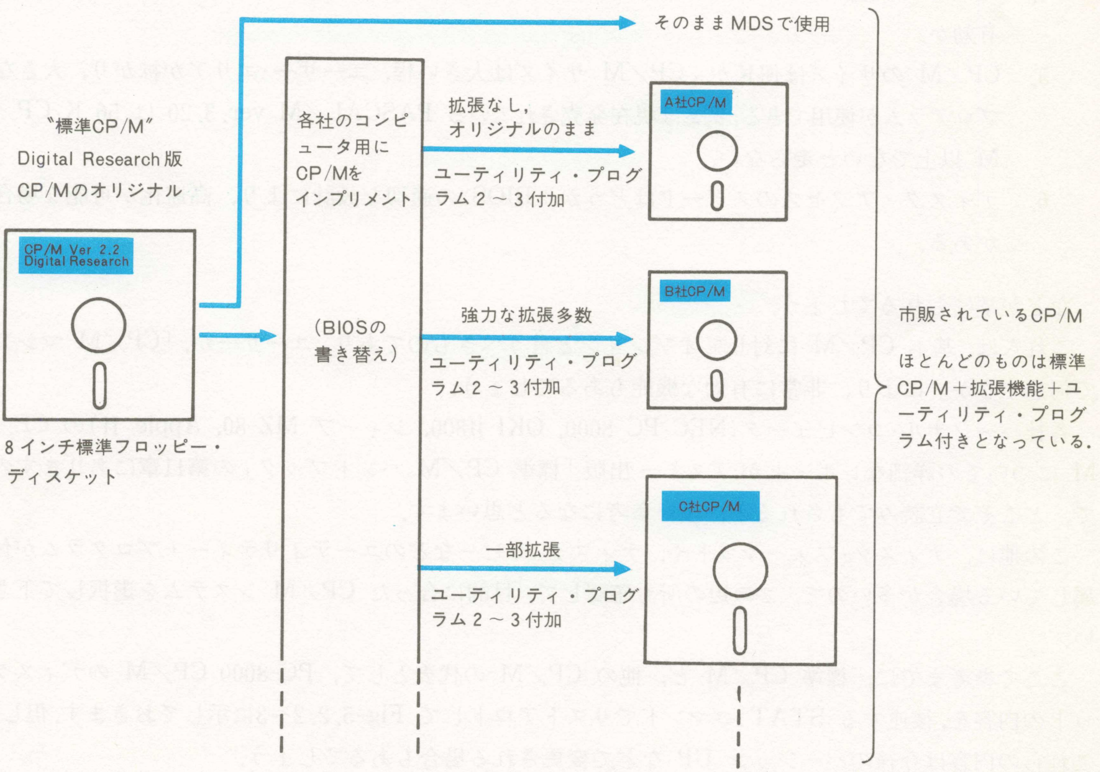


Figure-5.2.1 標準CP／Mから各マシンへのインプリメント

CP／M の基本部分は，どのシステムでもまったく同じなので，CP／M システムを選ぶ対象となるのは，BIOSに関する部分とそれに関するハードウェアということになります．BIOS(Basic I／O System)については3章の3.1にも解説してありますので参照して下さい．

ではユーザーに直接関係するそれらの項目を列記してみましょう．

1. ディスク・ドライブの形式：ミニであるか8インチであるか，片面・両面，単密・倍密，などの違いにより，記録容量に大きな差がある．
2. 周辺装置のサポートの状態：プリンタ，RS-232 C，IEEE-488，等などのポートがマシンに用意されていて，かつそれを CP／M がサポートしているか．
3. スクリーン・ディスプレイ関係で，カーソルのアドレッシングや，各種スクリーン・コントロールが，エスケープ・シーケンス★などで CP／M 上からコントロール可能か．
4. キーボード上に，ファンクション・キーなどの特種キーがあれば，それらは CP／M 上から有効か．
5. CP／M のサイズは何Kか：CP／M サイズは大きい程，ユーザー・エリアが拡がり，大きなプログラムが使用できる．例えば現在発売されている PASCAL／M ver.3.20 は，56 K CP／M 以上でないと走らない．
6. ディスク・アクセスのスピードはどうか：BIOS の適切な設計により，高速化が可能な場合がある．

などが挙げられるでしょう．

これらは，基本 CP／M に対してオプションと言うべきものであり，ユーザーが，「CP／M マシンで何をするか」により，非常に有効な機能もあるでしょう．

各社パーソナル・コンピュータ(NEC PC-8000，OKI if800，シャープ MZ-80，Apple II)の CP／M についての詳細なレポートが，アスキー出版「標準 CP／M ハンドブック」の第11章にありますので，どこぞで立読みでもされると，よい参考になると思います．

この他に，ディスク・フォーマットや，ディスク・コピーなどのユーティリティー・プログラムが付属している場合が多いので，この辺の所も確認して，目的に合った CP／M システムを選択して下さい．

ここで参考までに，標準 CP／M と，他の CP／M の代表として，PC-8000 CP／M のディスケットの内容を，後述する STAT コマンドでリストアウトして Fig-5.2.2～3に示しておきます．但し，これらの内容は今後のバージョン UP などで変更される場合もあるでしょう．

★ エスケープ・シーケンス――エスケープ・コード（1BH）の後に続く文字列で，カーソルのアドレッシングや，スクリーン表示のコントロールなど各種の操作を行うこと．

```
A>STAT B:*.*J
```

Recs	Bytes	Ext	Acc		
64	8k	1	R/W	B:ASM.COM	8080アセンブラ
96	12k	1	R/W	B:BIOS.ASM	BIOSの骨子のソース・プログラム
69	9k	1	R/W	B:CBIOS.ASM	MDS用BIOSのソース・プログラム
38	5k	1	R/W	B:DDT.COM	デバッガ
80	10k	1	R/W	B:DEBLOCK.ASM	ブロック／デブロックのソース・プログラム
49	7k	1	R/W	B:DISKDEF.LIB	自動的にディスク・ディファインを行なうライブラリ
33	5k	1	R/W	B:DUMP.ASM	ダンプ・プログラムのソース
4	1k	1	R/W	B:DUMP.COM	ダンプ・プログラム
52	7k	1	R/W	B:ED.COM	エディタ
14	2k	1	R/W	B:LOAD.COM	HEXファイル→COMファイル変換プログラム
76	10k	1	R/W	B:MOVCPM.COM	CP/Mシステムのリロケート・プログラム
58	8k	1	R/W	B:PIP.COM	周辺装置間のデータ転送プログラム
41	6k	1	R/W	B:STAT.COM	情況報告・状態設定用プログラム
10	2k	1	R/W	B:SUBMIT.COM	バッチ処理用プログラム
8	1k	1	R/W	B:SYSGEN.COM	システム生成用プログラム
6	1k	1	R/W	B:XSUB.COM	バッチ処理用の拡張プログラム

```
Bytes Remaining On B: 147k

A>
```

Figure-5.2.2　標準CP／Mの内容　(version 2.2)

```
A>STAT *.*
```

Recs	Bytes	Ext	Acc		
64	8k	1	R/W	A:ASM.COM	
7	1k	1	R/W	A:AUTHELLO.COM	
6	1k	1	R/W	A:AUTO-ST.COM	起動時にプログラムをオートスタートさせる機能をもつプログラム
18	3k	1	R/W	A:COPY.COM	バックアップ・コピー・プログラム
38	5k	1	R/W	A:DDT.COM	
49	7k	1	R/W	A:DISKDEF.LIB	
33	5k	1	R/W	A:DUMP.ASM	
3	1k	1	R/W	A:DUMP.COM	
52	7k	1	R/W	A:ED.COM	
10	2k	1	R/W	A:FORMAT.COM	
143	18k	2	R/W	A:HELLO.ASM	}特殊なハードウェアのI/O部分が記述されているファイル
32	4k	1	R/W	A:HELLO56.COM	
29	4k	1	R/W	A:KEY.COM	ファンクション機能をサポートするプログラム
14	2k	1	R/W	A:LOAD.COM	
32	4k	1	R/W	A:MODULE.HEX	
80	10k	1	R/W	A:MOVCPM.COM	
58	8k	1	R/W	A:PIP.COM	
41	6k	1	R/W	A:STAT.COM	
10	2k	1	R/W	A:SUBMIT.COM	
4	1k	1	R/W	A:SWITCH.COM	
7	1k	1	R/W	A:SYSGEN.COM	
66	9k	1	R/W	A:USER.ASM	}BIOSのI/O部分が記述されているファイル
101	13k	1	R/W	A:USERIO.ASM	
32	4k	1	R/W	A:WCLOCK.COM	デモンストレーション・プログラム
6	1k	1	R/W	A:XSUB.COM	

```
Bytes Remaining On A: 3k

A>
```

Figure-5.2.3　PC-8001 CP／Mの内容　(version 2.2, 81年9月現在)

5.3 CP/Mを買うと何が付いてくるのか？

CP／M を買うと（マシンではなく，ソフトウェアの方），そのパッケージには一体どんなものが入っているのでしょう．現物の写真で紹介しましょう．システムによっては，CP／M を走らせるために，特別のハードウェア・ボードを組合わせているものもあります．

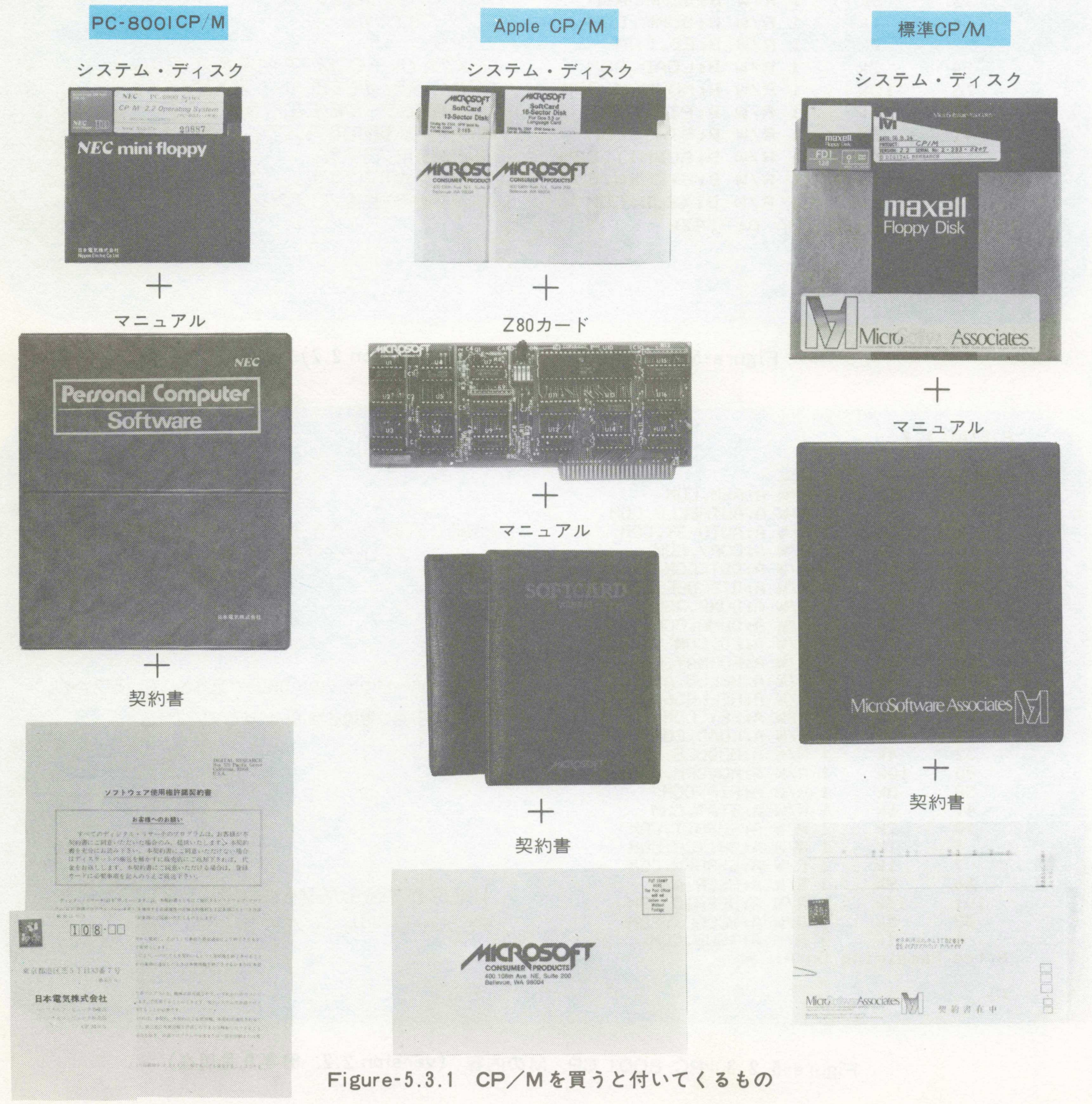
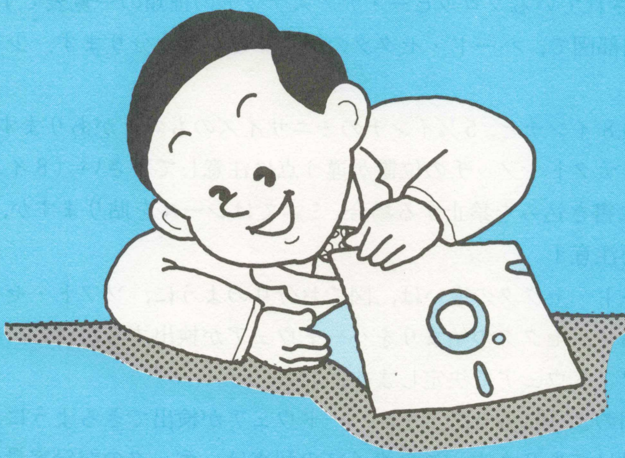


Figure-5.3.1 CP／Mを買うと付いてくるもの

6章　ディスクについての知識



フロッピー・ディスケットについて
ディスケットのバックアップの重要性
CP/Mはディスケット上にどのように記録されているか？

6.1 フロッピー・ディスケットについて

6.1.1 ディスケットの種類いろいろ

フロッピー・ディスクに使用するディスケットは，それぞれのディスク・ドライブに適合した種類のものを使用しなければなりません．例えば，8インチの両面用のディスケットは，片面ディスク・ドライブに挿入しても動作しませんが，その逆に，片面用ディスケットは，両面用ディスク・ドライブに挿入しても，片面使用では正常に使用できるのです．この項では，フロッピー・ディスケットについての一般的な解説をしておきましょう．

Fig-6.1.1は現在市販されているフロッピー・ディスケットの種類の一覧表です．Fig-6.1.2はそれぞれのディスケットの内部図で，ハード・セクタの構造などがよく分ります．少し詳しく解説しましょう．

大きさは標準サイズの8インチと，5¼インチのミニサイズのものとがあります．両者の間で書き込み禁止用のライト・プロテクト・ノッチの位置が違う点に注意して下さい（8インチではノッチのないものが多い）．おまけに書き込みを禁止する場合，ミニではシールを貼りますが，8インチでは逆に，シールをはがします．要注意！

ソフト・セクタ，ハード・セクタの違いは，図でお分りのように，ソフト・セクタでは，インデックス・ホールは1つであり，セクタの始まりをハードウェアが検出するだけで，続くセクタは，ディスク・コントローラがソフトウェアで決定します．

ハード・セクタは，図のように各セクタを，ハードウェアが検出できるように，セクタの数だけインデックス・ホールを設けてあります．このタイプの利点は，データの記録容量が増し，ディスク・コントローラの設計も簡単になるのですが，そのメリットも今や LSI によるディスク・コントローラの普及で意味がなく，かえっていろいろな面で使いにくく，設計が古いものでない限り最近では特殊なシステム用を除いては使われていません．

使用面は，片面と両面使用とがあり，片面の場合は，ラベルが貼ってある側の反対側の面を使用します．両面の場合は，前述の面を〝サイド0〟，ラベルの貼ってある側の面を〝サイド1〟と呼んで両サイドを使用します．記録容量は2倍になります．

記録密度は，単密度と倍密度とがあり，IBM 3740フォーマットである．1セクタ128バイトの密度に対して，1セクタ256バイトの2倍の記録密度にしたものを，倍密度（ダブル・デンシティー）と呼びます．但し，Fig-6.1.1にあるように1セクタ当りのバイト数は，1トラック当りのセクタ数により変ります．

		使用面	記録密度	レコード長（バイト）	セクタ	記録容量（Kバイト）
8インチ・ディスケット	ソフト・セクタ	片面	単密度	128	26	243
				256	15	280
				512	8	299
			倍密度	256	26	493
				512	15	568
				1024	8	606
		両面	単密度	128	26	493
				256	15	568
				512	8	606
			倍密度	256	26	985
				512	15	1137
				1024	8	1212
	ハード・セクタ	片面	単密度		32セクタ・ホール	315
			倍密度		32セクタ・ホール	631
		両面	単密度		32セクタ・ホール	631
			倍密度		32セクタ・ホール	1262
ミニ・ディスケット	ソフト・セクタ	片面	単密度	128	16	72
			倍密度	256	16	143
		両面	単密度	128	16	143
			倍密度	256	16	287
	ハード・セクタ	片面	単密度		10セクタ・ホール	90
					16セクタ・ホール	72
		両面	倍密度		10セクタ・ホール	358
					16セクタ・ホール	287

Figure-6.1.1　ディスケットの種類

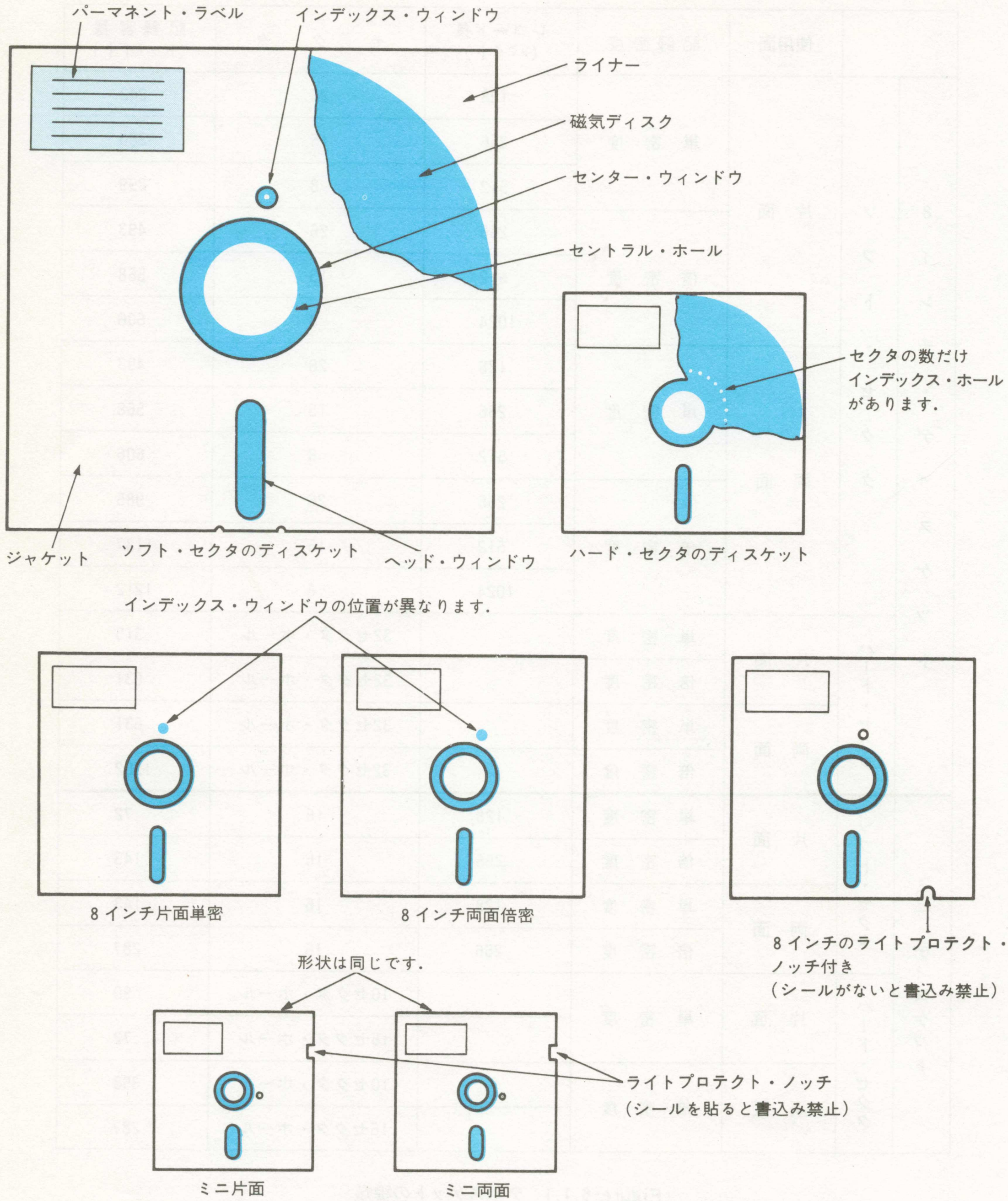


Figure-6.1.2 ディスケットの構造

6.1.2 ディスケットとドライブの適合

ハード・セクタのディスケットは，今日では一般的でないので省略し，ソフト・セクタのディスケットについて述べます．

ミニ・フロッピー・ディスケット

ミニの場合，形状については全種類共通です．片面用と両面用の差は，片面用は使わない側の品質が保障されていないこと，のみです．しかし現実には両者の製造工程の差は全くないらしく，片面用でまず支障なく両面用として使用できます．但し，反対側はあくまで品質保障の対象外ですので，エラーが発生しても文句は言えません．

単密と倍密に関しては，ミニの場合は区別せずに，始めから倍密の規格で製造しているメーカーがほとんどのようで，どちらにもまず問題なく使用可能です．

ミニに関して注意しなくてはならないことは，フォーマットの規格が統一されていないので，使う前に必ず使用するドライブに合ったフォーマット★(イニシャライズ)をしなければならないことです．

8インチ・フロッピー・ディスケット

まず，両面ドライブで両面使用として使用する場合は，両面用ディスケットでなければ使えません．両面ドライブでも，片面ドライブとして使用する場合は，片面用ディスケットが使えます．

単密と倍密に関しては，規格に差があるメーカーが多いので，倍密は倍密用のものを使った方がよいでしょう．

フォーマットに関しては，片面単密用ディスケットは，特殊なものを除いて，IBM 3740フォーマットでイニシャライズされていますので，通常のドライブならば，そのままで即使用できます．

6.1.3 ディスケットの耐久性について

常識的な環境での使い方ならば1つのトラックに付き1000万パス(ヘッドが接触して通過する回数)以上使用可能と言われており，これはプロッピー・ディスクが，何かの機械の一部として，連日連夜動作するとしても，何年も使える値です．一般のユーザーが，ソフトウェアの開発に使う程度なら〝永久に使える〟と言った方が当っているほど長持ちするでしょう．

8インチのドライブは，電源が入っていれば，アクセスに関係なく常に回転していますが，ジャケットの中でリード／ライト ヘッドが触れずに回転している分なら，何十日，何百日続けても平気です．但し，ジャケットの中に，金属片，砂，プラスチック片などの異物が入り込むと非常に短時間で完全なるダメージを受けますので，使用環境にはくれぐれも配慮して下さい．

★ フォーマット――ディスケットに基本的なアドレス情報などを書き込む処理のこと (8.6参照)．

6.2 ディスクのバックアップの重要性

まず第1に心得ておかなければならないことは，ディスクの〝バックアップ〞の必要性です．

ディスケットに記録されているデータは，常に次のような危険にさらされています．

1. 人為的ミス・オペレート
2. バグのあるプログラムの暴走．
3. ディスケットに対する物理的損傷．
4. ハードウェアのトラブル．

などにより，どれ程貴重なデータであっても一瞬のうちに〝パー〞になる危険性が常にあると言うことを，よーく頭に入れておいて下さい．〝パー〞になることを，〝クラッシュ〞と言います．ソフトウェア・ハウスの人達はよく〝トばした〞などと表現しています．いずれにしても，ディスクは常に〝クラッシュ〞の悲劇を背負っていることを認識しなくてはなりません．

これはマイクロ・コンピュータでも，大型コンピュータでも同じことで，記録ファイルのバックアップは非常に重要なことなのです．それを図で示したのが Fig-6.2.1です．

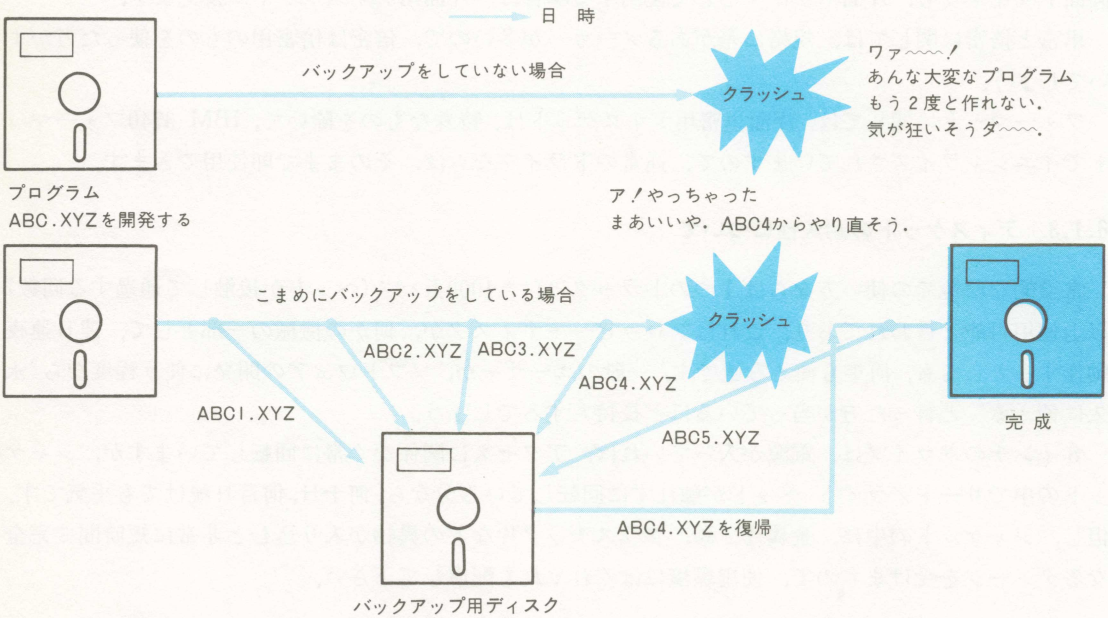


Figure-6.2.1 バックアップの重要性

今述べていることは，ディスク・システムの信頼性がないと言う意味ではありませんので誤解なさらないように．安価なパーソナル・コンピュータのディスク装置であっても，人が〝悪いこと〟をしなければ，その信頼性は非常に高いのです．

この件についての対策は，バックアップ・コピーを作ることと，ミス・オペレートに注意することが基本です．その他にもデバッグ中のプログラムの実行には，必要のないディスク・ドライブは，ふたを開けておき物理的に書き込めない状態にして，中のディスケットを暴走によるクラッシュから守る，というようなテクニックも必要になってくるでしょう．しかし何はともあれ，手間を惜しまずこまめにバックアップ・コピーを作ることが一番なのです．

大切なファイルは，実際の仕事には決して使わない習慣をつけ，別のディスクにコピーしてから，それを使いましょう．

6.3 CP/Mはディスケット上にどのように記録されているか？

まず，我々が〝CP／M〟と呼んでいるディスケットの内容（購入した CP／M ディスケットの内容）は，大きく分けて，２つの部分から成っています．

1) CP／M 自身＝CP／M システム（CP／M 本体）
2) CP／M に付属の各プログラム（PIP，ED，STAT，など★）

CP／M 本体のことを，ソフトウェアであっても〝システム〟と呼びます．本書では，〝CP／M システム〟と呼ぶことに統一しますが，この場合は，ハードウェアの〝装置〟の意味ではありませんのでご注意下さい．上記1)のCP／M システムが起動のための操作により，ディスクからコンピュータのメモリにロードされ，働き出すことを CP／M が〝起動した〟と言います．

CP／M システムに内蔵されているコマンド以外の CP／M プログラム（コマンド）や，ユーザー・プログラムなどは，CP／M システムには組込まれていない〝システム〟外のプログラムによって実行されます．

さて〝CP／M〟のディスケットには，これら２つの部分が記録されているわけですが，具体的にはどこに書き込まれているのでしょう．８インチ片面単密度の標準ディスケットを例に，解説してみましょう．両面ドライブやミニなどの場合も考え方は同じです．

Fig-6.3.1はディスケットの裏側（ラベルの貼ってない側）の図です．ディスケットの磁性面に同心円のトラックが77本あります．オーディオ・レコード盤のようなスパイラル状にはなっていません．ディスクの外周から内周に向けてトラック No. が付けてあり，外周側からトラック０，１，２，……76

★ PIP，ED，STAT，など――Fig-5.2.2参照．

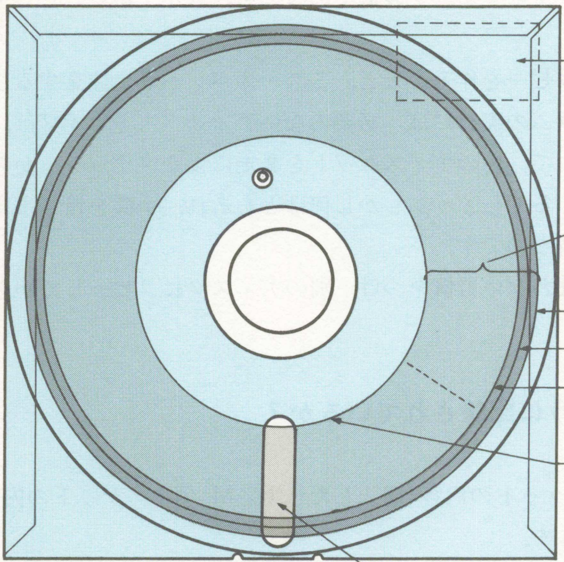


Figure-6.3.1 ディスケットの内部

まで計77本あります.

このトラック本数は，ディスク・ドライブの種類によって異り，ミニ・ディスクの場合は35本とか40本程度になっています.

CP／M はこれらのトラックに次のように書き込まれているのです.

トラック０～１　CP／M システム．ビルトイン・コマンド★を実行するためのプログラムはこの中に含まれている.（起動時には，この２本のトラックのデータが読み出されて，メモリにロードされ CP／M が起動する）

トラック２～76　トランジェント・コマンド★の各プログラムやその他のファイルとディレクトリ.（のちほどユーザーにより各種のファイルとそのディレクトリが書き込まれることになる）ディレクトリ（各ファイルの住所録）は，トラック２に書き込まれる.

★ ビルトイン・コマンド――システムに組込まれているコマンド．９章参照.

★ トランジェント・コマンド――呼ばれた時にディスクからメモリ上に読み出されてから実行されるコマンド．11章参照.

〝ディレクトリ〟と言うのは，ディスケット上に記録された各種ファイルの〝住所録〟とでも理解しておいて下さい．このディレクトリの記録されている最初のセクタであるトラック02，セクタ01の内容をシンクウェア・ラブズ社のユーティリティ・プログラムのセクタ・ディスプレイ・プログラム(SECDIS)で見てみましょう．Fig-6.3.2に示します．〝住所録〟が記録されている状態がよく分ります．1つのファイルに対して32バイトが割当てられていますね．

このようにして〝CP／M 全体〟はディスケット上に書き込まれています．今後，ディスケットのコピーや，ファイルのコピーなどを行う場合，ここでの知識があると，たいへん役に立つでしょう．

```
A>SECDIS↵

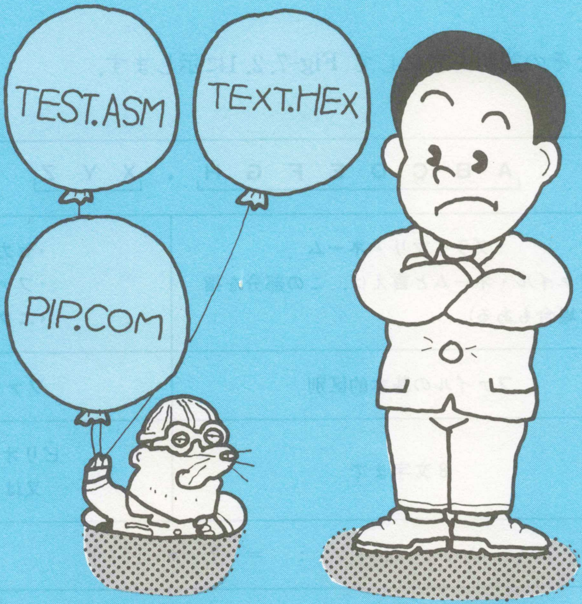
     ======   SECTOR DISPLAY PROGRAM  V1.5   ======
              copyright by SYNCWARE LABS

input disk name A - D  >B
input track# 00-76  >02
input sector# 01-26  >01
D)isplay,   N)ext sector disp.,  A)uto next,   X) any sector   R)eboot CP/M
input   D,  N,  A,  X,  R,   >D

DISK=B  TRACK=02   SECTOR=01
B:00  00 4D 4F 56 43 50 4D 20 20 43 4F 4D 00 00 00 4C  .MOVCPM  COM...L
B:10  02 03 04 05 06 07 08 09 0A 0B 00 00 00 00 00 00  ................
B:20  00 50 49 50 20 20 20 20 20 43 4F 4D 00 00 00 3A  .PIP     COM...:
B:30  0C 0D 0E 0F 10 11 12 13 00 00 00 00 00 00 00 00  ................
B:40  00 53 55 42 4D 49 54 20 20 43 4F 4D 00 00 00 0A  .SUBMIT  COM....
B:50  14 15 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
B:60  00 58 53 55 42 20 20 20 20 43 4F 4D 00 00 00 06  .XSUB    COM....
B:70  16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Figure-6.3.2 〝SECDIS〟によるディスケットの内容表示

7章　ファイルとファイル名



ファイルとは？
ファイル名
特別の意味を持つエクステンション

7.1　ファイルとは？

CP／M 上で，ディスクにデータを書き込んだり，読み出したりする動作はすべて〝ファイル〟を基に行われます．〝ファイル〟には必ず〝ファイル名〟が付いており，CP／M はユーザーから与えられた（或いは実行するプログラムがプログラムの中で指定する）ファイル名を頼りに，そのプログラムを実行したり，ファイルをタイプアウトしたり，その他いろいろな動作を行うのです．

7.2　ファイル名

ファイル名の形式とその説明を表にして Fig-7.2.1に示します．

書　式	A B C D E F G H	・ X Y Z
呼び方	・プライマリ・ネーム （ファイル・ネームと言えば，この部分を指す場合もある）	・セカンダリ・ネーム ・ファイル・タイプ ・エクステンション
用　法	ファイルの基本的区別	ファイルの種類の区別
文字数	8文字まで	ピリオドの後3文字まで 又はなし（付けない）
使用禁止文字	< > . , ; : = ? * [ ]	
ファイル名についての注意	○カナ文字が使用できるCP/Mでも，ファイル名だけにはカナの使用はできない． ○ファイル名に英小文字は使用できるが，小文字として認識はされない．すべて大文字とみなされる． ○ファイル名の間にブランクを置いてはいけない． ○ファイル名の全くないファイルも作ることは可能であるが，実際には使わない方がよい．	

Figure-7.2.1　ファイル名

次に実際のファイル名がどんなものか，後ほど解説する〝STAT〟プログラムでリスト・アウトしてみましょう．例としては，本書の実習で〝システム・ディスケット〟として使用するCP/Mディスケットに含まれる全てのファイルをリスト・アウトしてFig-7.2.2に示します．リスト右側の枠で囲んだ部分がファイル名です．左側には，ファイルの長さなどが表示されていますが，それらの解説は後章にして，ファイル名の〝エクステンション〟に注目して下さい．

DUMP.ASM

DUMP.COM

などと，プライマリ・ネームは同じで，エスクテンションが異るものがありますね．この3文字以内のエスクテンションはユーザーにより，自由に付けられるものですが，CP/Mにとって特別の意味を持つものがいくつかありますので，それらについて解説しましょう．

```
A>STAT *.*↲

 Recs  Bytes  Ext Acc
   64     8k    1 R/W A:ASM.COM
   96    12k    1 R/W A:BIOS.ASM
   69     9k    1 R/W A:CBIOS.ASM
   38     5k    1 R/W A:DDT.COM
   80    10k    1 R/W A:DEBLOCK.ASM
   49     7k    1 R/W A:DISKDEF.LIB
   33     5k    1 R/W A:DUMP.ASM
    4     1k    1 R/W A:DUMP.COM
   52     7k    1 R/W A:ED.COM
   14     2k    1 R/W A:LOAD.COM
   76    10k    1 R/W A:MOVCPM.COM
   58     8k    1 R/W A:PIP.COM
  222    28k    2 R/W A:PTBIOS48.ASM
   26     4k    1 R/W A:PTBOOT48.ASM
   68     9k    1 R/W A:PTCPM48.COM
   41     6k    1 R/W A:STAT.COM
   10     2k    1 R/W A:SUBMIT.COM
    8     1k    1 R/W A:SYSGEN.COM
    6     1k    1 R/W A:XSUB.COM
Bytes Remaining On A: 106k

A>
```

Figure-7.2.2　本書の実習で使用するシステム・ディスクの内容

7.3 特別の意味をもつエクステンション

CP／M にとって特別の意味とはどういうことなのか, まず上記の〝DUMP.ASM″, 〝DUMP.COM″の 〝ASM″ と 〝COM″ の2つのエクステンションについて解説しましょう.

ASM……これは CP／M のアセンブラのソース・ファイルに必ず付けるエクステンションです.

ソース・ファイルである 〝DUMP.ASM″ の内容を, 後ほど実習する 〝TYPE″ コマンドでタイプアウトして, Fig-7.3.1に示します. CP／M アセンブラは, エクステンションが〝ASM″であるファイルに限りアセンブル可能であり, 〝ASM″以外はソーフ・ファイルとみなさず, アセンブルしません.

```
A>TYPE DUMP.ASM↲

;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;       COPYRIGHT (C) 1975, 1976, 1977, 1978
;       DIGITAL RESEARCH
;       BOX 579, PACIFIC GROVE
;       CALIFORNIA, 93950
;
        ORG     100H
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF   EQU     15      ;FILE OPEN
READF   EQU     20      ;READ FUNCTION
;
FCB     EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
BUFF    EQU     80H     ;INPUT DISK BUFFER ADDRESS
          .
          .
          .
          .
          .
```

Figure-7.3.1 アセンブラ・ソース・ファイル

COM……これは CP／M 上で即実行可能な純マシン・オブジェクト・ファイルに必ず付けるエクステンションです.

〝COM″ エクステンションの付いているファイルは, そのプライマリ・ネームのみキーインすることにより, ディスクからメイン・メモリにロードされて, 即実行されます.

```
A>DUMP DUMP.COM
0000 21 00 00 39 22 15 02 31 57 02 CD C1 01 FE FF C2
0010 1B 01 11 F3 01 CD 9C 01 C3 51 01 3E 80 32 13 02
0020 21 00 00 E5 CD A2 01 E1 DA 51 01 47 7D E6 0F C2
0030 44 01 CD 72 01 CD 59 01 0F DA 51 01 7C CD 8F 01
0040 7D CD 8F 01 23 3E 20 CD 65 01 78 CD 8F 01 C3 23
0050 01 CD 72 01 2A 15 02 F9 C9 E5 D5 C5 0E 0B CD 05
0060 00 C1 D1 E1 C9 E5 D5 C5 0E 02 5F CD 05 00 C1 D1
0070 E1 C9 3E 0D CD 65 01 3E 0A CD 65 01 C9 E6 0F FE
  .
  .
  .
  .
```

Figure-7.3.2 オブジェクト・ファイル

参考までにオブジェクト・ファイルの〝DUMP.COM〟を，ダンプ・プログラムである自分自身でダンプして，Fig-7.3.2に示します．これは逆アセンブルすると（〝DDT〟プログラムで行う．後述）Fig-7.3.3に示すようになり，Fig-7.3.1の後に続くソース・ファイルと一致します．

```
-L100,500
  0100  LXI  H,0000
  0103  DAD  SP
  0104  SHLD 0215
  0107  LXI  SP,0257
  010A  CALL 01C1
  010D  CPI  FF
  010F  JNZ  011B
  0112  LXI  D,01F3
  0115  CALL 019C
  0118  JMP  0151
  011B  MVI  A,80
  011D  STA  0213
    .
    .
    .
    .
```

Figure-7.3.3 逆アセンブル・リスト

このようにエクステンションには，特別の意味を持つものがあり，それらの内，よく使われるものを表にして Fig-7.3.4に示します．

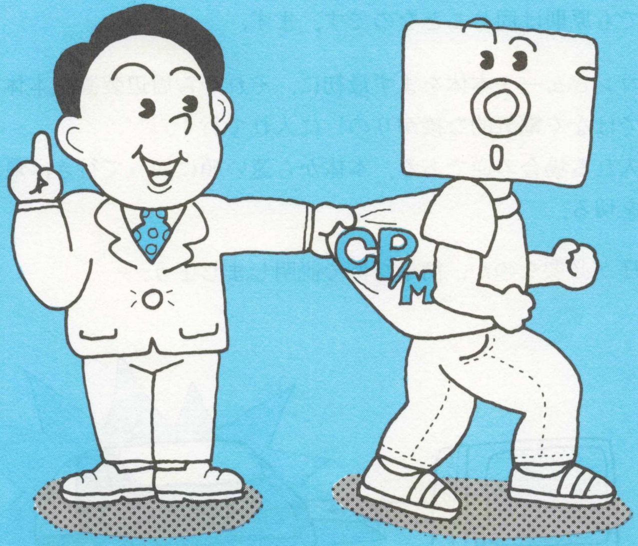
エクステンション	それぞれのエクステンションが付いたファイルの意味.
.ASM	アセンブル・ソース・ファイル. エディタによりユーザーが作成する.
.COM	即実行可能オブジェクト・ファイル. トランジェント・コマンド・ファイルと言う. "LOAD" プログラムによってHEXファイルから生成される.
.HEX	インテル・HEX 形式のオブジェクト・ファイル. アセンブル(ASM)・プログラムにより生成される.
.BAK	バックアップ・ファイル. エディタ(ED)操作により, オリジナル・ファイルを保存するために自動的に作成される.
.PRN	アセンブル・ソース・ファイルをアセンブルした後のリスティング・ファイル. アセンブル(ASM)プログラムにより自動的に生成される.
.LIB	ライブラリ・ファイルを表わす. エディタ(ED)において, メイン・ファイルと結合が可能. ユーザーが作成する.
.SUB	"SUBMIT" プログラムによりバッチ処理するための各種コマンドをまとめたファイル. ユーザーが作成する.
.$$$	テンポラリ・ファイル. EDなどを実行する時, 一時的に作成され, 実行が完了すれば削除される, プログラム内部で必要な一時的ファイル.

Figure-7.3.4 エクステンション一覧表

特別な意味を持つエクステンションは, ここに挙げたもの以外にもあり, 例えば, 高級言語の場合は, MBASIC では〝BAS〟, COBOL-80では〝COB〟などと, それぞれのソース・ファイルには決ったエクステンションを付けなければなりません.

しかし, このような特別の意味のあるファイル以外のものは, どのようなエクステンションを付けようと自由です. 後日になっても識別が容易な名前を付ければよいのです.

8章　さあCP/Mを走らせよう，その前に



コンピュータ・システムの電源を入れる前に
CP/Mの起動
プロンプト記号
ミスタイプ時の1文字デリート
リブート（ウォーム・スタート）とコールド・スタート
ディスケットのフォーマッティング
ディスケットのバックアップ・コピーの作り方
CP/Mのエラーについて

8.1 コンピュータ・システムの電源を入れる前に

まず，コンピュータ・システムの電源を入れる場合と切る場合についての常識について話さなければなりません．

コンピュータ本体と CRT ディスプレイとカセットという簡単な組合わせの場合は，気にする必要はありませんが，周辺装置に，プリンタ，ディスク・ドライブ，モデムなどの通信装置や，被制御機器など種々のものが接続されている場合は，電源の ON-OFF には注意しなければなりません．これは大型コンピュータでも原則は同じことなのです．まず，

電源を入れる時：コンピュータ本体をまず最初に，それから周辺装置を本体に近い順（物理的距離ではなく電気的な接がりの）に入れて行く．

切 る 時：入れる場合の逆である．本体から遠い順に切って行き，最後にコンピュータ本体を切る．

なぜこのような順序が必要なのか，簡単な例で説明しましょう．

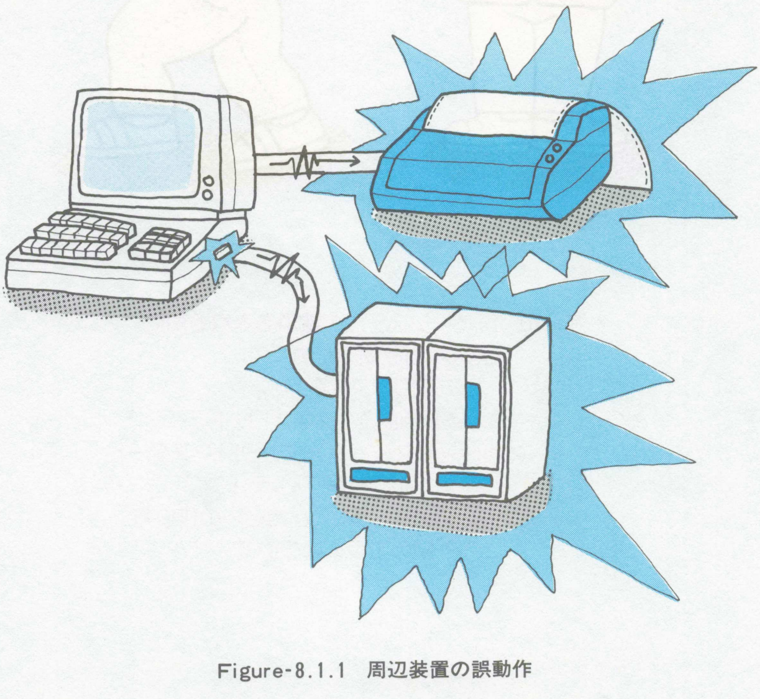


Figure-8.1.1 周辺装置の誤動作

例えば Fig-8.1.1のように，コンピュータ本体に，プリンタが接続されていたとします．そしてプリンタの電源はすでに ON になっているとしましょう．つまりプリンタは，印字指令が来れば即動作する状態にあるわけです．

そこでコンピュータの電源を ON したり OFF したりします．どうなるでしょう．何も起らないかも知れません．しかし何回か繰返すうちにきっとプリンタが，何らかの動作を起すことがあるでしょう．これがコンピュータ本体の電源の ON-OFF 時に発生する不安定な状態による周辺装置の誤動作の一例です．

これがプリンタであれば，不用な文字が印字される程度なのでまだよいのですが，ディスク・ドライブであった場合などは，重大な結果を招くことになります．

システムの電源 ON-OFF には，この原則に従い周辺装置の誤動作，暴走を防がねばなりません．但し，コンピュータ・システムによっては，本体と周辺装置が同時に ON になるものもあり，また，大抵のシステムは原則に反したことを行っても，誤動作が起らないよう一応の対策は施してあるでしょう．しかし本項の主旨は理解しておく必要があります．

8.2 CP/Mの起動

コンピュータ本体や周辺装置の電源はすでに ON になっているとします．CP／M を起動させる方法は，各コンピュータによって異りますが，ほとんどの CP／M マシンでは，CP／M のシステム・ディスケット（CP／M 自身が記録されているディスケット）をドライブAに挿入し，リセット・ボタンを押すか，あるいは IPL（イニシャル・プログラム・ローダ）ボタンを押すことにより，自動的に CP／M が起動します．しかしコンピュータによっては，簡単な操作手順を必要とするものもありますので，各々のコンピュータの操作マニュアルに従って下さい．

オープニング・メッセージ

CP／M が起動すると必ずオープニング・メッセージが表示されます．そしてオープニング・メッセージは，最低限，次の３項目から成っています．

1. ″CP／M″ というプログラム・タイトル
2. Version No.　(1.4，2.2 など)
3. CP／M のメモリサイズ★（60K，58K，44Kなどいろいろ）

★ メモリサイズ――Fig-11.1参照．大きい程ユーザー・エリアも大きい．

例えば，一番簡素なオープニング・メッセージは次のようなものです．

```
58K CP/M  vers 2.2
A>
```

Figure-8.2.1 CP／Mオープニング・メッセージの一例

この例では，バージョン2.2で，サイズ58 K の CP／M が起動したことを表わしています．

8.3 プロンプト記号〝A＞〞

CP／M が起動すると，オープニング・メッセージに続いて，必ず〝A＞〞が表示されます．これは CP／M の〝プロンプト〞と言って，「CP／M のコンソール・コマンド★を受け付けます」という意味を表わす記号です．〝A＞〞のAは，現在のログイン・ディスクがドライブAであることを示しています．CP／M ではディスク・ドライブ名をA，B，C，D，……P（16ドライブまで増設可能）と呼び，起動時は必ずAがログイン・ディスクになります．

ログイン・ディスク（Log-in disk）は一般の DISK BASIC ではなかった考え方で，カレント・ディスクとも言い，９章の「ログイン・ディスクのチェンジ」の項で解説しますが，複数個あるディスク・ドライブの中で，現在，ベースになっているドライブ，あるいは現在自分のものになっているドライブのことを指す，と思って下さい．

★ コンソール・コマンド――ビルトイン・コマンドやトランジェント・コマンド（プログラム）などを働かせるためのコマンド．

8.4　ミス・タイプ時の１文字デリート

実習に当り，とりあえずキーインした文字を１文字デリートする方法を覚えましょう．

DEL（デリート） または CTRL－H（コントロール） または RUB（ラブアウト） または BS（バック・スペース）

どのようなコンピュータでも，デリート操作には上の４種類のキーのうち，どれか一つは使えるはずです．最近のコンピュータのキーボードは，大抵は〝DEL〞キーを備えているので，このキーでミス・タイプした文字をデリートできます．デリートされた文字は，順次 CRT の表示から消えて行きます．

コンピュータによっては，CRT から消えず，逆にデリートされた文字を再表示（エコー・バックと言う）するものもあります．

例えば，



とキーインして，E，D，Cの３文字をデリートした場合，エコー・バックされるスクリーン上の表示は Fig-8.4.1のようになります．

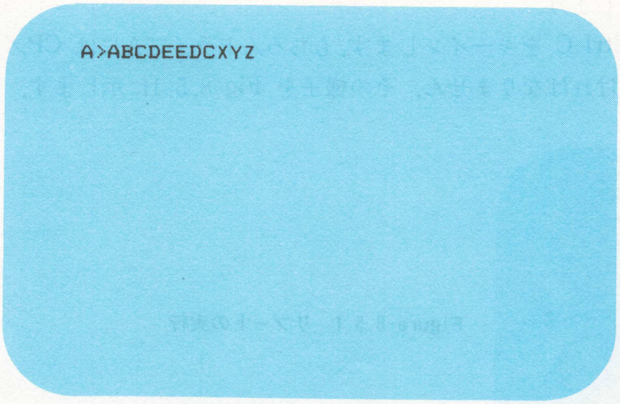


Figure-8.4.1　１文字デリート
（エコー・バック形式）

これはきれいに書き直すと次のようになります．

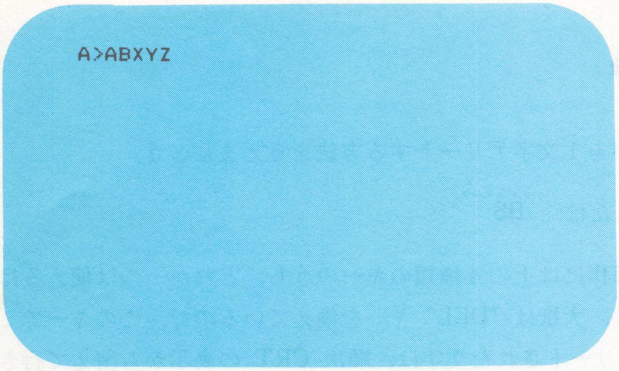


Figure-8.4.2 デリート後の結果

きれいに書き直すのは Ctrl-R で行えますが，これらは10章の「ライン・エディティング機能」の項で述べます.

いずれにせよ，前述の４種類のいづれかのキー操作で，１文字デリートが可能です．あなたが使用するコンピュータの CP／M マニュアルを読み，実際に試して，使い易いキーを使用して下さい.

8.5 リブート（ウォーム・スタート）とコールドスタート

CP／M を運用する上で〝リブート〟(Reboot) 操作は欠くことのできないものであり，「CP／M」と言えば〝リブート〟を連想するほど多用するオペレーションです.

とりあえず実際にリブートを実行してみましょう.

キーボードからリブートを起させるには Ctrl-C をキーインします．もちろんドライブAには CP／M のシステム・ディスク★が挿入されていなければなりません．その様子を Fig-8.5.1に示します.

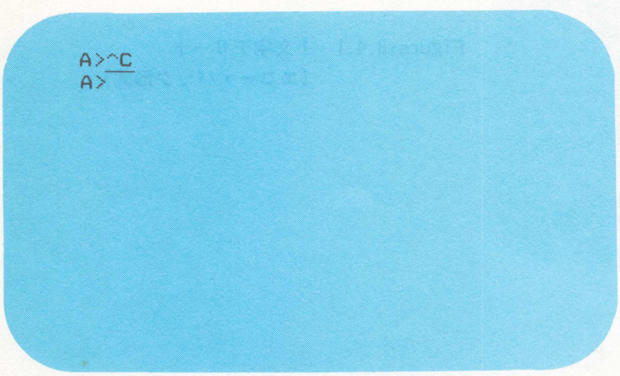


Figure-8.5.1 リブートの実行

★ システム・デスク――CP／M の〝システム〟が記録されているディスク．６章を参照.

〝∧C〃は Ctrl-C をキーインしたスクリーン上の表示であり，CP／M ではコントロール・キャラクタを入力すると，頭に〝∧〃記号を付けてコンソールに表示します．

Ctrl-C をキーインした時，何が起りましたか？

ドライブAがアクセスされる音がして，再度〝A＞〃が表示されたでしょう．この時，CP／M システムの一部がディスクから再ロードされ，かつ CP／M 内部がソフトウェア的にリセットされたのです．リセット・ボタンや IPL による CP／M の起動（コールド・スタート）と異なる点は，すでにメモリにロードされているユーザー・プログラムを破壊しないことと，CP／M の BIOS 部などは，再ロードされず以前のままであることです．

このリブートを行う目的などを，リセット・ボタンや IPL による起動（コールド・スタート）と対比してまとめたものが Fig-8.5.2です．

名　　称	動　　作	使用目的
(リセット・ボタン又はIPLによる起動) コールド・スタート 又はコールド・ブート	外部のローダの力を一時貸り，ディスクからCP/Mシステムをメイン・メモリにロードして，CP/Mを起動させる．	全くのゼロからCP/Mを起動し，全てをイニシャライズする．
(ctrl-C 又は0Hスタートによる) リブート 又はウォーム・スタート 又はウォーム・ブート	すでにメイン・メモリにロードされているCP/Mシステムを使用して，ディスクからシステムの一部を再ロードし，CP/M各部をイニシャライズする．この場合，すでにロードされているユーザー・プログラムを破壊することはない． キーボードからはCtrl-Cにより行う．	すでにログインされたディスケットを別のディスケットと交替した場合，自動的に書き込み禁止となる．これを書き込み可能にするために新たにログインし直す． 実行中のトランジェント・プログラムを打切り，又は終了させて，CP/Mにもどるための“ブレーク”として用いる． トランジェント・プログラムによって壊されたCCP部を再ロードする．

Figure-8.5.2　リブートとコールド・スタートの比較

リブートはキーボードからは Ctrl-C で起りますが，ユーザー・プログラムからはアドレス0H にジャンプさせる〝JMP 0H〃を実行することにより起ります．アドレス0000H には常に，リブートのためのジャンプ・ベクトルが書かれているからです．

8.6 ディスケットのフォーマッティング

CP／M マシンで使用する新しい空のディスケットは，それぞれの CP／M に付属しているフォーマット・プログラムを実行して，それぞれのディスク・ドライブに適合したフォーマット（イニシャライズ）の作業を行わなければ使用することはできません．

ディスケットを購入したままの状態で，どのマシンにもそのまま使用できるのは，8インチ片面単密度の〝標準ディスケット〟と呼ばれているものだけです．6章でも述べましたが，このフォーマットが IBM 3740であり，各ソフトウェア会社がディストリビュートするメディアの代表は，このディスケットなのです．8インチ片面単密度のドライブを使用している CP／M マシンは，必ずこの IBM フォーマットを採用しており，市販されているこのタイプのディスケットも，IBM フォーマットでイニシャライズしてから出荷されているため，ユーザーによるフォーマットを必要としないのです．その他のタイプのディスケットはフォーマットの規格が統一されていなかったり，CP／M に合致しなかったりするので，ほとんどのものはユーザーがフォーマットする必要があります．

〝フォーマット〟の意味は，直接 CP／M と関係があるわけではありません．フォーマットを直接必要とするのは，ディスク・ドライブと，ディスク・コントローラなのです．ディスク・ドライブが，ディスク・コントローラから命令される，〝任意のトラックの任意のセクタのデータ〟を正確に読み書きするために必要な〝ディスク上のアドレス情報〟を前もってディスクに書き込んでおくことを，ディスケットの〝フォーマット〟とか〝イニシャライズ〟とか呼んでいるのです．

フォーマットによりディスケットに書き込まれたアドレス情報は，ファイルのセーブや，イレーズなどのファイル操作では決して消去することはできません．よってフォーマットの作業は一度行えば，あとは事故でフォーマットの部分までもクラッシュしない限り，必要はありません．但し，ディスケット上のデータをすべて消去して〝きれいな〟ディスケットにするためには，しばしばこのフォーマットを代用します．

フォーマット・プログラムのほとんどは実行すると，前述のアドレス情報と共に，ディスケットのデータ部全面に，16進の〝E5〟のコードが書き込まれます．CP／M はこの〝E5〟のコードを，ディレクトリを格納する，専用のトラックでのみ利用していますが，その他のトラックでは全く関係ありません．（8インチ両面倍密のディスケットは購入時にはE5ではなく40が書き込まれている）

参考までに，CP／M が記録されているディスケット（ドライブA側）と，フォーマットをした直後のディスケット（ドライブB側）の一番最初の，トラック00，セクタ01に記録されている内容を見てみましょう．9章でも使用した SECDIS プログラムを使いました．その様子を Fig-8.6.1に示します．

〝アドレス情報〟に関しては残念ながら，ディスク・コントローラの内部だけで処理されるものであり，我々の前には，このセクタのディスプレイのようには姿を現してくれません．

```
A>SECDIS↲

     ======   SECTOR DISPLAY PROGRAM  V1.5   ======
               copyright by SYNCWARE LABS

input disk name A - D  >A
input track# 00-76  >00
input sector# 01-26  >01
D)isplay,   N)ext sector disp.,   A)uto next,   X) any sector   R)eboot CP/M
input   D,  N,  A,  X,  R,   >D

DISK=A  TRACK=00   SECTOR=01
A:00  1E 0A 31 00 01 21 00 A4 16 33 0E 02 06 04 79 CD  ..1..!...3....y.
A:10  2A 00 15 CA 00 BA 06 00 0C 79 FE 1B DA 0F 00 3E  *........y.....>
A:20  53 D3 E8 DB EC 0E 01 C3 0C 00 D3 EA CD 41 00 3E  S............A.>
A:30  88 B0 D3 E8 DB EC B7 F2 41 00 DB EB 77 23 C3 34  ........A...w#.4
A:40  00 DB E8 E6 9D C8 1D C2 02 00 32 80 00 2F D3 FF  ..........2../..
A:50  C3 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .P..............
A:60  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
A:70  00 00 00 00 00 00 00 00 00 00 00 00 00 C3 00 00  ................

D)isplay,   N)ext sector disp.,   A)uto next,   X) any sector   R)eboot CP/M
input   D,  N,  A,  X,  R,   >X
input disk name A - D  >B
input track# 00-76  >00
input sector# 01-26  >01
D)isplay,   N)ext sector disp.,   A)uto next,   X) any sector   R)eboot CP/M
input   D,  N,  A,  X,  R,   >D

DISK=B  TRACK=00   SECTOR=01
B:00  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  ................
B:10  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  ................
B:20  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  ................
B:30  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  ................
B:40  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  ................
B:50  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  ................
B:60  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  ................
B:70  E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  ................

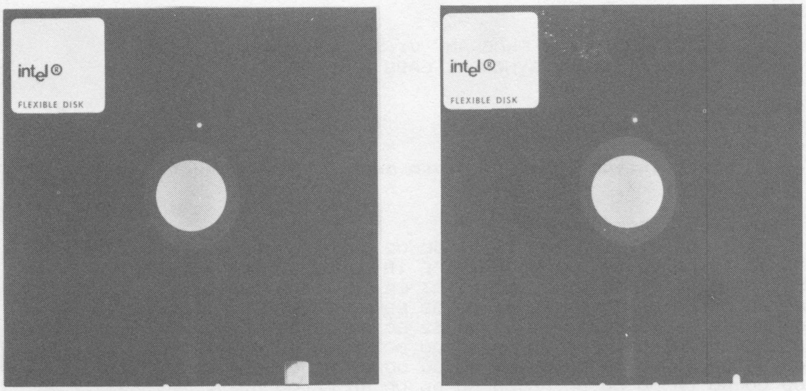
D)isplay,   N)ext sector disp.,   A)uto next,   X) any sector   R)eboot CP/M
input   D,  N,  A,  X,  R,   >R
A>
```

Figure-8.6.1　フォーマット直後のディスケットの内容

8.7　ディスケットのバックアップ・コピーの作り方

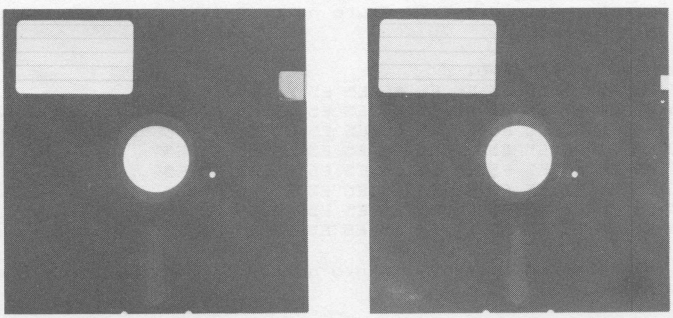
CP／M の実習を始める前に，何はともあれ CP／M のシステム・ディスケットのバックアップ・コピーを作らねばなりません．オリジナル・ディスケットは，コピーを作る時のみ使用する程度にして，常時は大切に保管しておきましょう．ライト・プロテクト・ノッチ付きのディスケットならば，オリジナルのものは書き込み禁止にしておいた方が安心です．シールの貼り方を Fig-8.7.1に示します．

書込み可能 ➡ 禁　止



8インチ・ディスケットの場合

禁　止 ➡ 書込み可能



ミニ・ディスケットの場合

ミニと8インチではシールを貼る貼らないは全く逆．要注意！

Figure-8.7.1　ライトプロテクト・ノッチとシール

ではコピーの作業を始めましょう．コピーの対象は，購入したばかりの自分のコンピュータ用のCP／Mディスケット，という想定です．コピー作業は，もしコピー専用のユーティリティー・プログラムが付属していれば，それを使った方が簡単で速く行えますが，ここではCP／Mのコマンドを使って行う例を示します．リストの下線部がキーインの部分，⏎はキャリッジ・リターンを示します．

1）オリジナル・ディスケットをドライブA，新ディスケットをドライブBに挿入し，CP／Mを起動する．

2）オリジナル・ディスケットに含まれているすべてのファイル名を，〝DIR〟コマンドであらかじめ確認しておく．サンプルランをFig-8.7.2に示します．

```
A>DIR↲
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: BIOS     ASM : CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB
A: PTBIOS48 ASM : PTBOOT48 ASM : PTCPM48  COM
A>
```

Figure-8.7.2 DIRコマンドによるシステム・ファイルの確認

3）新ディスケット（もちろん使い古しのディスケットであってもよい）を，付属のディスク・フォーマット・プログラムでフォーマット（イニシャライズ）する．フォーマット・プログラムは，CP／M マシンにより少しづつ違いがあるので，それぞれのマニュアルに従う．いずれにしろ，フォーマットされる側のドライブをよく確認し，くれぐれもオリジナルのディスケットをフォーマットしてしまうことのないように．

Fig-8.7.3にフォーマット・プログラムのサンプルランを示します．

```
A>FORMAT↲

DISK INITIALIZATION PROGRAM

BLANK DISK READY TO SET ON DRIVE B: ? (Y / N) >Y
FUNCTION COMPLETE

BLANK DISK READY TO SET ON DRIVE B: ? (Y / N) >N
A>
```

Figure-8.7.3 〝FORMAT〟によるディスケットのイニシャライズ

4）現在，ドライブAにはオリジナル・ディスケット，ドライブBにはフォーマットされた空ディスケットが挿入されています．

次は，CP／M のシステム部を〝SYSGEN〟プログラムを使って空ディスケットにコピーします．そのサンプルランを Fig-8.7.4に示します．プログラムの問いに対する，A又はBのキーインを間違えないように．

最後の問いに，リターンを入力すれば，この例のように CP／M にもどりますが，再び〝B〟を入力すれば，B側のディスケットを取り替えながら，何枚でもシステム部のコピーが行えます．

```
A>SYSGEN↲
SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)A
SOURCE ON A, THEN TYPE RETURN↲
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B
DESTINATION ON B, THEN TYPE RETURN↲
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)↲

A>
```

Figure-8.7.4 "SYSGEN"によりCP／Mシステム部を作る

5）システム部のコピーが終ったら，次はオリジナル・ディスケットに記録されているすべてのファイルを"PIP"プログラムを使ってコピーします．そのサンプルランを Fig-8.7.5に示します．もし何らかのエラー表示が出力された場合は，Ctrl-C をキーインしてリブートするか，あるいはリセット・ボタンによりコールド・スタートして，もう一度 PIP を実行して下さい．

```
A>PIP B:=*.*↲

COPYING -
MOVCPM.COM
PIP.COM
SUBMIT.COM
XSUB.COM
ED.COM
ASM.COM
DDT.COM
LOAD.COM
STAT.COM
SYSGEN.COM
DUMP.COM
DUMP.ASM
BIOS.ASM
CBIOS.ASM
DEBLOCK.ASM
DISKDEF.LIB
PTBIOS48.ASM
PTBOOT48.ASM
PTCPM48.COM

A>
```

Figure-8.7.5 "PIP"によりすべてのファイルをコピーする

6）これで，ドライブAのオリジナル・ディスケット上の記録は，すべてドライブBのディスケット上にコピーされました．確認のため，新しくコピーされたディスケットをドライブAに挿入し，リセット・ボタンを押して起動させてみましょう．起動できたら次にb）と同様に〝DIR〟コマンドでファイル名を一応確認しておきます．

以上でコピーの作業は終りです．ここで使用した CP／M のコマンドやプログラムは，後程解説しますので，ここではただ使えるだけで結構です．

8.8 CP/Mのエラーについて

CP／M を使っていると，時折り

BDOS　ERROR　ON　A：R／O

とか，

OUTPUT　FILE　WRITE　ERROR

などのいくつかのエラー・メッセージが出力されることがあります．大抵の場合は，次に示す項目のいずれかですので，よく確認した上で処理して下さい．

1）起動又はリブートした後で，ディスケットを入れ替えたり，フタを開け閉めし，ディスクへの書き込みを行うコマンドを実行した時．
この場合，CP／M はディレクトリの破壊を防ぐために自動的に書き込み禁止になる．
処置――Ctrl-C でリブートするか，最初から起動を行う．

2）ディスケット上の空きスペースがないのに，書き込みを行なった時．
処置――後述の STAT プログラムで，空きスペースを確認し，不用なファイルを削除するか，別のディスク上に書き込みを行う．

3）ディレクトリが満杯（ファイルの数が制限いっぱい）になっているのに，さらに新しいファイルをセーブしようとした時．
処置――一応 DIR コマンドで，ファイルの状態を調べ，不用のファイルを削除するか，別のディスケット上にセーブするようにする．

4）存在するディスク・ドライブ以外のドライブ名を誤ってコマンドなどで指定した時．

上記の原因などで，エラー・メッセージが出力され，処理が中断された時など，リターン・キーを入力することによりエラーを無視して処理を続行することも可能な場合がありますが，そのようなことは避け，Ctrl-C でリブートしてコントロールをもどすのがよいでしょう．

9章　CP/Mの使い方，ビルトイン・コマンド基礎実習



ビルトイン・コマンドとは？
実習のためのディスケットの準備
DIR(ファイル名リスト・アウト・コマンド)
TYPE(ファイル内容タイプ・アウト・コマンド)
x:(ログイン・ディスクのチェンジ)
ERA(ファイル削除コマンド)
REN(ファイル名変更コマンド)
SAVE(メモリ内容のディスク・セーブ・コマンド)
USER(ユーザー・ナンバー指定コマンド)

9.1　ビルトイン・コマンドとは？

CP／M は第6章でも述べたように，〝CP／Mシステム〟と，CP／M に付属の各プログラムのファ.イルから成っています．そして，CP／M システムに内蔵されているコマンドを〝ビルトイン・コマンド〟と言い，システム外の各プログラムによるコマンドを，〝トランジェント・コマンド（プログラム）〟と呼んでいます．

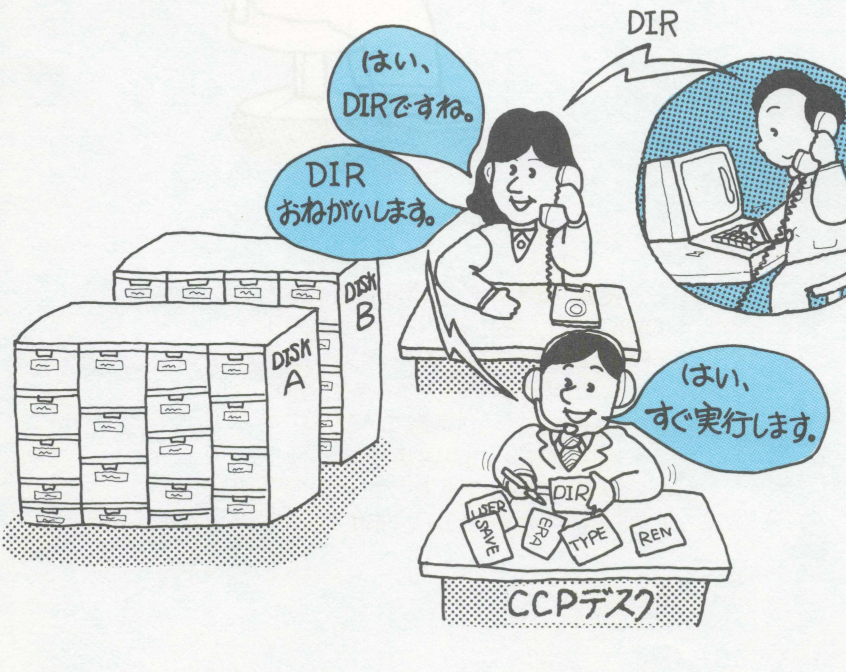
CP／M が起動すると，CP／M システムはメイン・メモリのアドレスの上位の方にロードされ，ビルトイン・コマンドもシステムと一体のため当然同時にロードされています．

つまり，ビルトイン・コマンドを実行するためのプログラムは，CP／M が起動していれば RAM に常駐しており，呼ばれれば即実行することができる状態にあるわけです．

これに対し，トランジェント・コマンド（プログラム）は，呼ばれた時，まずディスクからそのプログラムを読み出して，メインメモリにロードしてから実行されるのです．

本書では，各コマンドの基本的な使い方を実習します．まずはCP／Mに慣れることが必要であり，各コマンドのさらに細部にわたる実習は続巻である〝実習 CP／M〟で行われます．

リストの下線部がキーインする部分，♪はキャリッジ・リターンを示します．



9.2 実習のためのディスケットの準備

7章で作った CP/M システム・ディスケットのコピーをドライブAに挿入し，他のディスケットで CP/M 用のファイルが，いくつか入っているものがあれば，それをコピーして，ドライブBに挿入します．そのようなディスケットがない場合は，システム・ディスケットをもう一枚コピーして，ドライブBにも同じものを挿入しておけばよいでしょう．

実習に使用する CP/M マシンのディスクが，1ドライブである場合でも，ドライブAに関する実習の部分だけを行えば大部分は理解できるでしょう．

A，B両ドライブにディスケットが挿入されたら，それぞれのマシンによる方法で CP/M を起動して下さい．大抵のマシンは，リセットボタンかあるいは IPL ボタンにより CP/M が起動します．

CP/M が起動してプロンプト（〝A＞〞）が出力されたのを確認したら，実習に入ります．

各項目のタイトルになっているコマンド名の下に書かれた呼び方は，一般的なものを示しましたが，人により様々に呼ばれるものもあります．

9.3 DIR（ファイル名リスト・アウト・コマンド）

ディーアイアール

―― DIRectory コマンド ――

機能　任意のディスクに含まれるファイル名をリストアウトする．

実習A　ログイン・ディスク★に含まれるすべてのファイル名をリスト・アウトする

Fig-9.3.1がそのサンプルランで，〝A＞〞で示されているように，ドライブAが現在のログイン・ディスクであり，そこに挿入されているディスケットの全部のファイル名がリスト・アウトされます．

```
A>DIR↲
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: BIOS     ASM : CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB
A: PTBIOS48 ASM : PTBOOT48 ASM : PTCPM48  COM
A>
```

Figure-9.3.1　ログイン・ディスクのすべてのファイル名を見る

★ ログイン・ディスク――現在のベースになっているディスク・ドライブ（9.5参照）．

実習B　任意のディスクに含まれるすべてのファイル名をリスト・アウトする

Fig-9.3.2がそのサンプルランで,現在ログインしていないドライブに挿入されているディスケットのファイル名全部をリスト・アウトします．サンプルランでは〝B：〞によりドライブB上のファイルをリスト・アウトしています．同様に〝C：〞,〝D：〞などとP：まで16のドライブを指定することができます（それらのドライブがあれば）．

```
A>DIR B:⏎
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: LIB      COM : CREF80   COM : TESTPRO  BAS : FORLIB   REL
B: MBASIC   COM
A>
```

Figure-9.3.2　Bドライブのすべてのファイル名を見る

実習C　任意のファイル名を任意のディスクから捜し出し，見つかればリスト・アウトする

Fig-9.3.3のサンプルランは，ログイン・ディスクのファイルの中から，〝DUMP.ASM〞を捜しています．これは存在していました．

```
A>DIR DUMP.ASM⏎
A: DUMP     ASM
A>
```

Figure-9.3.3　〝DUMP.ASM〞を捜す

Fig-9.3.4のサンプルランは，ログイン・ディスクのファイルの中から，〝ABC.XYZ〞を捜しています．これは存在しないという答が返ってきました．

```
A>DIR ABC.XYZ↲
NO FILE
A>
```

Figure-9.3.4 〝ABC.XYZ〞を捜す

Fig-9.3.5のサンプルランは，ログイン・ディスク以外の任意のディスクのファイルの中から指定したファイルを捜します．この例では，ドライブBの中から，〝MBASIC.COM〞を捜しています．

```
A>DIR B:MBASIC.COM↲
B: MBASIC    COM
A>
```

Figure-9.3.5 Bドライブの〝MBASIC.COM〞を捜す

実習D　任意のディスクのファイルのうち，ある条件にマッチするもののみを捜し出してリスト・アウトする

Fig-9.3.6のサンプルランはログイン・ディスク内の，ファイル名のエクステンションが〝ASM〞であるすべてのファイル名をリスト・アウトします．〝*〞はプライマリ・ネームやエクステンションの代りに使用し，〝すべての〞という意味を持つファイル・マッチ記号です．

```
A>DIR *.ASM↲
A: DUMP     ASM : BIOS     ASM : CBIOS    ASM : DEBLOCK  ASM
A: PTBIOS48 ASM : PTBOOT48 ASM
A>
```

Figure-9.3.6　エクステンションに〝ASM〞を持つファイル名をリスト・アウトする

Fig-9.3.7のサンプルランは，エクステンションにファイル・マッチ記号〝*〞を使った例です．プライマリ・ネームが〝DUMP〞であるすべてのファイル名がリスト・アウトされます．

```
A>DIR DUMP.*↲
A: DUMP     COM : DUMP     ASM
A>
```

Figure-9.3.7　プライマリ・ネームに〝DUMP〞を持つファイル名をリスト・アウトする

Fig-9.3.8のサンプルランは，ファイル・マッチ記号に〝?〞を使った例です．〝?〞は，その位置でのすべてのキャラクタの1文字に相当します．この例では〝????〞の4文字の部分が何んであ

ってもよいわけです.

```
A>DIR PT????48.*↲
A: PTBIOS48 ASM : PTBOOT48 ASM
A>
```

Figure-9.3.8 ファイル・マッチ記号（?）の使用例

Fig-9.3.9のサンプルランは,ログイン・ディスク以外の任意のディスクのファイル名にもファイル・マッチが使えることを示しています. この例ではドライブBの中から, エクステンションが"COM"であるファイルのすべてをリスト・アウトしています.

```
A>DIR B:*.COM↲
B: BASCOM   COM : M80      COM : L80      COM : LIB      COM
B: CREF80   COM : MBASIC   COM
A>
```

Figure-9.3.9 Bドライブのファイルに対してファイル・マッチ（*）を使用

DIR注 各種ファイルには11.2章で解説しますが"STAT"プログラムにより,いくつかのファイル・アトリビュート（ファイル属性）を付けることができます. 例えば "$SYS" アトリビュートを付けたファイルは DIR コマンドでは発見することが出来ず, そのファイルは"ない"とみなされます. しかし $SYS などの機能は MP／M★を使う上で必要なものであり, CP／M ではほとんど必要ありません. よってここでは触れていません.

★ MP／M———マルチユーザー用の CP／M（13章参照）.

9.4 TYPE（ファイル内容タイプ・アウト・コマンド）

タイプ

—— TYPE out コマンド ——

機能 ログイン・ディスク，または任意のディスク上の任意のアスキー・ファイルをコンソールにディスプレイする．

実習A ログイン・ディスク内のアスキー・ファイルをコンソールにディスプレイする

Fig-9.4.1のサンプルランは，現在ログイン・ディスクであるドライブA上のファイル〝DUMP.ASM〟（ダンプ・プログラムのソース・ファイル）をタイプ・アウトします．

```
A>TYPE DUMP.ASM↲

;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;       COPYRIGHT (C) 1975, 1976, 1977, 1978
;       DIGITAL RESEARCH
;       BOX 579, PACIFIC GROVE
;       CALIFORNIA, 93950
;
        ORG     100H
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF   EQU     15      ;FILE OPEN
READF   EQU     20      ;READ FUNCTION
;
FCB     EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
BUFF    EQU     80H     ;INPUT DISK BUFFER ADDRESS
        .
        .
        .
        .
```

Figure-9.4.1 〝DUMP.ASM〟をタイプ・アウト

ファイルの最後までタイプ・アウトされると，自動的に CP／M にもどります．又，タイプ・アウトの動作中に何らかのキー入力をすると，ブレークがかかり，その時点で CP／Mにもどります．
表示を一時的にポーズ状態にするには，Ctrl-S をキーインします．

実習B　任意のディスク上のアスキー・ファイルをディスプレイする

Fig-9.4.2のサンプルランは，ドライブBにある MBASIC のソース・ファイル″TESTPRO．BAS″をタイプ・アウトします．

```
A>TYPE B:TESTPRO.BAS↲

10 '============================================================
20 '
30 '          ITI   CS   UCS   FOR MOUSE
40 '
50 '============================================================
60 '
70 PRINT "===== DISCRIMINATED AVOIDANCE-SAME PROGRAM WITH 4 BOXES =====
80 PRINT
90 '
100 '***** INITIAL SET *****
110 '
120 DEFINT A-Z     'DEFAIN ALL VALIABLES TO INTEGER
130 '
140 'DISK FILE NAME DEFINE
150 INPUT "FILE NAME";FILENAMEIN$
160 INPUT "DATE";DATEIN$
170 '
180 'DISK FILE OPEN
190 OPEN "R",#1,FILENAMEIN$
200 '
210 '--- DATA STORE AREA DIM DEFINE ---
220 'DATA BYTE BIT ASSIGNMENT
      .
      .
      .
      .
```

Figure-9.4.2　Bドライブの″TESTPRO.BAS″をタイプ・アウト

実習C アスキー・ファイル以外はタイプ・アウトできません.

Fig-9.4.3に即実行可能なオブジェクト・ファイルの "DUMP.COM" を TYPE コマンドでタイプ・アウトしたサンプルランを示します. アスキー・コードでなければ文字や記号にならないのでこのようにメチャメチャの表示しか出力されません. TYPE コマンドはアスキー・ファイルのみタイプ・アウトが可能です.

```
A>TYPE DUMP.COM↲
!9"1WMA^BsMCQ>2!eM"aZQ6}{BDMrMYZQ!M}M#> MexMC#Mr$yIeUE
MMQaIeUE_MAQaI>
MeIf^
RFOC
F7MeIuM)qM}I          MI:~B3MN7J37I_<2!~7I/2I¥MIeUE¥MAQaIFILE DUMP VERSION 1.4$
MOJ<NPUT FILE PRESENT ON DISK$j!uEUeYMaQAqI~ H~ H~,H~
MMM$JA!^
@T@
MMC':   M$Jr#
^ IVO^
X
```

Figure-9.4.3 アスキー・ファイル以外のファイルのタイプ・アウト

9.5　x：（ログイン・ディスクのチェンジ）

CP／M の起動時は，必ずドライブAがログイン・ディスクとなり，プロンプトは〝A＞〟と出力されます．しかし，CP／M を運用する上で，ログイン・ディスクをA以外の任意のドライブに変更したい場合が生じます．ログイン・ディスクを変更する必要性などは，この時点で解説するのは困難ですが，実習が進むにつれて，自然と理解されてくるでしょう．

とりあえず次のように理解しておいて下さい．ログイン・ディスクと言うのは，〝自分〟のディスクであり何かプログラムを実行する場合，特別にドライブ名の指定をしなければ，通常はすべて自分のディスク上だけで，各種の処理が行われます．

それに対し，ログイン・ディスク以外のディスクは，〝他人〟のディスクであり，それらを使う場合には必ず〝お宅のものを使いますョ〟というお断りのドライブ名記号（A：，B：，C：，……）が必要になります．

また，処理によっては，〝自分〟のディスク内になければプログラムの実行が行えないものもあり，その他，オペレーション上の都合などで，A，B，C，……の各ディスク上を，〝自分〟のディスクを求めて渡り歩く必要性があるわけです．

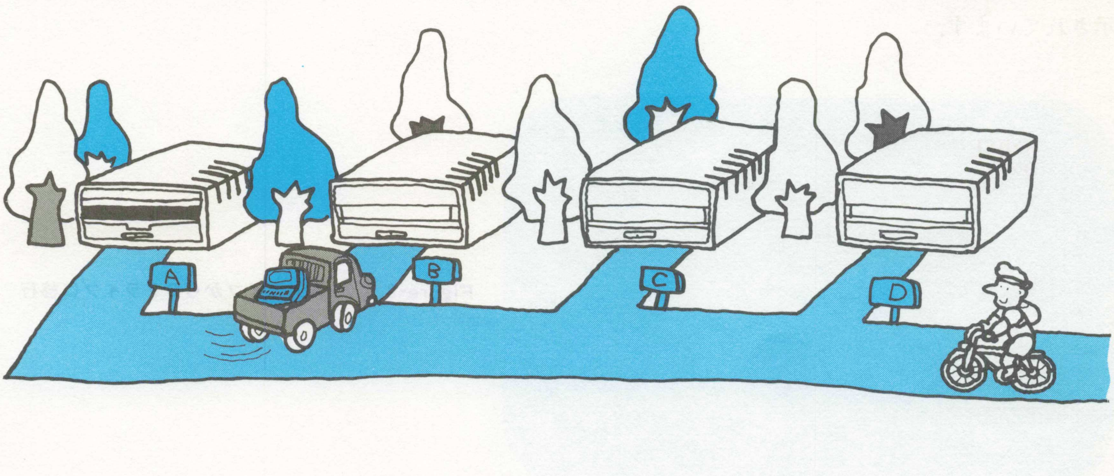


Fig-9.5.1は9.3章でも示したように，ログイン・ディスクがAの状態で，ディスクBのディレクトリを調べるサンプルランです．コマンドに〝B：〟が付いています．これは参考のためのサンプルランで，のちほど比較します．

```
A>DIR B:↵
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: LIB      COM : CREF80   COM : TESTPRO  BAS : FORLIB   REL
B: MBASIC   COM
A>
```

Figure-9.5.1 Bドライブの内容を見る

Fig-9.5.2がチェンジ・ディスクのサンプルランで，この場合は，ドライブAからドライブBに移行しています．〝ドライブ名に：（コロン）を付けてリターン〟これで任意のドライブにログインすることができます．但し，新しくログインするドライブには必ずディスケットが挿入されていなければなりません．

この例ではBに移行したので，ログイン・ディスクがBであることを示すプロンプト〝B＞〟が表示されています．

```
A>B:↵
B>
```

Figure-9.5.2 AドライブからBドライブに移行

Fig-9.5.3は,ログイン・ディスクがBの状態で DIR コマンドを実行したサンプルランです.〝自分〟のディスク上のファイル名がリスト・アウトされており, Fig-9.5.1と同一内容です. このように, 目的のファイルが存在するデイスクにログインすれば, ドライブを指定する〝x:〟を付けなくてもよいわけです.

```
B>DIR↲
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: LIB      COM : CREF80   COM : TESTPRO  BAS : FORLIB   REL
B: MBASIC   COM
B>
```

Figure-9.5.3 Figure-9.5.1と同じである

Fig-9.5.4は再び元のドライブAにログインするサンプルランです. プロンプト〝A>〟が表示され, Aがログイン・ディスクになりました.

```
B>A:↲
A>
```

Figure-9.5.4 再びAドライブに移行

今行ったことを，TYPE コマンドで示せば Fig-9.5.5～6になります．この２つのコマンドの書き方は，目的は同じなのですが，前者はログイン・ディスクはそのままでタイプ・アウト，後者はドライブＢに移行してからタイプ・アウトを行っています．もちろん結果は同じです．

```
A>TYPE B:TESTPRO.BAS↲
```

Figure-9.5.5　Aドライブから B ドライブ上の "TESTPRO.BAS" をタイプする

‖　　結果は同じです．

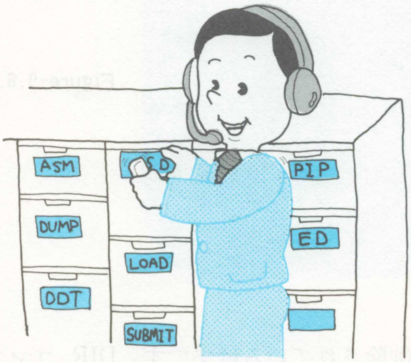
```
A>B:↲
B>TYPE TESTPRO.BAS↲
```

Figure-9.5.6　ドライブＢに移行して "TESTPRO.BAS" をタイプする

9.6 ERA（ファイル削除コマンド）
イレーズ

—— ERAse コマンド ——

機能　　任意のディスク上の任意のファイルを削除する.



注）ERA コマンドの実習では，ディスクのファイルを消してしまうので，あらかじめもう一枚コピーを作っておいて下さい．コピーの方法は，8.7章を見て下さい．

実習A　任意のディスク上の１つのファイルを削除する

まずログイン・ディスク内の１つのファイルを削除します.

その前に一応全部のファイルを DIR コマンドでリスト・アウトして内容の確認をしましょう．それを Fig-9.6.1に示します.

```
A>DIR↲
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: BIOS     ASM : CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB
A: PTBIOS48 ASM : PTBOOT48 ASM : PTCPM48  COM
A>
```

Figure-9.6.1 ディスク内容の確認（ERA実行前）

この中のファイル，〝DISKDEF . LIB〞を削除します．そのサンプルランを Fig-9. 6. 2に示します．

```
A>ERA DISKDEF.LIB↲
A>
```

Figure-9.6.2 ERAコマンドにより〝DISKDEF. LIB〞を削除

これで〝DISKDEF . LIB〞は削除されているはずです．DIR コマンドで確認します．それを Fig-9. 6. 3に示します．

```
A>DIR↲
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: BIOS     ASM : CBIOS    ASM : DEBLOCK  ASM : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48  COM
A>
```

Figure-9.6.3 ディスク内容の確認（ERA実行後）

このように削除されていることがわかります．

実習B　任意のディスク上のある条件にマッチしたファイルを削除する

ERAコマンドは，ファイル・マッチ記号（*，?）が使えます．Fig-9. 6. 4では，ログイン・ディスク内のすべての〝COM〞エクステンションの付いたファイルを削除します．

```
A>ERA *.COM
A>
```

Figure-9.6.4　ファイル・マッチ（*）を使って削除

すべての〝COM〟ファイルが削除されたか，DIR で確認してみましょう．Fig-9.6.5に示します．

```
A>DIR
A: DUMP     ASM : BIOS     ASM : CBIOS    ASM : DEBLOCK  ASM
A: PTBIOS48 ASM : PTBOOT48 ASM
A>
```

Figure-9.6.5　ディスク内容の確認

このようにすべての〝COM〟ファイルが削除されています．

実習C　任意のディスク上のすべてのファイルを削除する

ログイン・ディスク上のすべてのファイルを削除してみましょう．これは特別なコマンドを使うわけではなく，ファイル・マッチの応用なのですが，事が重大なだけに，CP/M が心配して，「ほんとに全部消していいのか？」と確認を求めてきます．〝Y〟をキーインし，リターンすれば ERA コマンドが実行され，〝N〟をキーインし，リターンすれば何も起らず CP/M にもどります．そのサンプルランを Fig-9.6.6に示します．

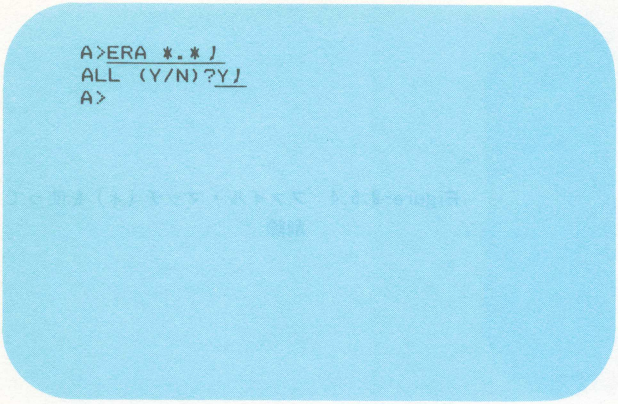


Figure-9.6.6 すべてのファイルの削除

すべてのファイルが削除されているか, DIR コマンドで確認してみましょう. Fig-9.6.7に示します.

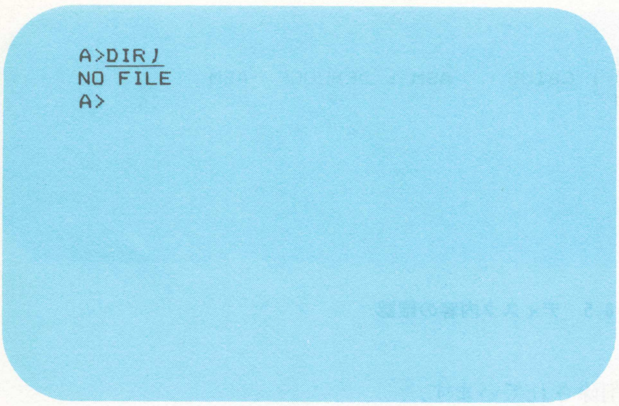


Figure-9.6.7 ディスク内容の確認

このように，ファイルは何も存在しないことが確認されます.

ここで誤解してはいけないことは，ERA コマンドで削除できるのは〝ファイル〞であり，〝ファイル〞ではない〝CP／M システム〞は削除できないことです．よって，Fig-9.6.6ですべてのファイルを削除しても〝システム〞はそのまま残っているわけです．リセット・ボタンか，IPL ボタンなどで，全部のファイルを消されたディスクの CP／M を起動してみて，システムが消去されてないことを試して下さい.

実習D　ログイン・ディスク以外のファイルを削除する

今まで，ログイン・ディスク内のファイルを対象に ERA コマンドの実習を行いましたが，ログイン・ディスク以外のドライブ上のファイルも全く同様に削除することができます．コマンドの形式は DIR コマンドと同じく，ファイル名の前にドライブ名とコロン (A：，B：，C：，……) を付けるだけです．

サンプルランとして，ドライブB上のファイル"FORLIB.REL"をAにログインしたままで削除してみます．一応 DIR コマンドでドライブBのファイルを確認し，ERA コマンドを実行してみましょう．以上を Fig-9.6.8と Fig-9.6.9に示します．

```
A>DIR B:↲
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: LIB      COM : CREF80   COM : TESTPRO  BAS : FORLIB   REL
B: MBASIC   COM
A>
```

Figure-9.6.8　Bドライブの内容を見る

```
A>ERA B:FORLIB.REL↲
A>
```

Figure-9.6.9　Bドライブの"FORLIB.REL"を削除

ドライブＢ上の〝FORLIB．REL〞が削除されているか，DIR コマンドで確認してみましょう．Fig-9.6.10に示します．

```
A>DIR B:↲
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: LIB      COM : CREF80   COM : TESTPRO  BAS : MBASIC   COM
A>
```

Figure-9.6.10 Ｂドライブの内容の確認

このようにログイン・ディスク以外のファイルでも削除可能です．ファイル・マッチやオール削除についても同様です．

ERAコマンド運用上のアドバイス

大抵のコマンドがそうであるように，ERA コマンドも Fig-9.6.9のように，ログインされているディスク以外のドライブにもアクセスすることができます．しかしこの方法は思わぬ〝うっかり〞で，正しい〝x：〞を指定しなかったり，〝x：〞をキーインしたつもりが抜けていたりして，消してはいけないものを消してしまう危険性があります．

そこで，安全のためには，ログイン・ディスクを目的のファイルが存在するドライブにチェンジしてから ERA コマンドを実行することをおすすめします．そして実行する前には，DIR コマンドでファイル全体を見て，確かに目的のファイルであることを確認しましょう．

Fig-9.6.11～12にログイン・ディスクをチェンジして，ドライブＢ上の〝LIB．COM〞を削除するサンプルランと，実行後の DIR コマンドによるリスト・アウトを示します．

```
A>B:↲
B>ERA LIB.COM
B>
```

Figure-9.6.11 目的のドライブに移行してから削除

```
B>DIR↲
B: BASLIB    REL : BASCOM    COM : M80       COM : L80       COM
B: CREF80    COM : TESTPRO   BAS : MBASIC    COM
B>
```

Figure-9.6.12 実行後の確認

9.7 REN（ファイル名変更コマンド）
リネーム

—— REName コマンド ——

機能　任意のディスク上のファイル名を変更する.



（注. 前項の ERA コマンドの実習でファイルを消してしまったので, 新たにコピーしたディスケットを, ドライブA, Bに挿入し, 再度 CP／M を起動しておくこと）

実習A　ログイン・ディスク上の任意のファイル名を変更する

ログイン・ディスク（現在はA）上には Fig-9.7.1に示すファイルがあります.

```
A>DIR↲
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: BIOS     ASM : CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB
A: PTBIOS48 ASM : PTBOOT48 ASM : PTCPM48  COM
A>
```

Figure-9.7.1　ログイン・ディスクの内容を見る

この中のファイル"BIOS.ASM"を"ABCDEFG.XYZ"に変更してみましょう．サンプルランをFig-9.7.2に示します．

```
A>REN ABCDEFG.XYZ=BIOS.ASM↲
A>
```

Figure-9.7.2　リネームの実行

上の例のようにコマンド・ラインは"＝"の左辺が新ファイル名，右辺が旧ファイル名になります．

REN　新ファイル名＝旧ファイル名

コマンドの形式の右辺・左辺を間違えないように覚えて下さい．

さて，"BIOS.ASM"が"ABCDEFG.XYZ"とリネームされているか，DIR コマンドで確認してみましょう．Fig-9.7.3に示します．

```
A>DIR↲
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : DUMP     ASM
A: ABCDEFG  XYZ : CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB
A: PTBIOS48 ASM : PTBOOT48 ASM : PTCPM48  COM
A>
```

Figure-9.7.3　リネーム後のファイル名の確認

確かに"ABCDEFG.XYZ"に変更されています．

参考までに旧"BIOS.ASM"，今は"ABCDEFG.XYZ"を TYPE コマンドで内容をタイプ・アウトしてみましょう．Fig-9.7.4に示します．

```
A>TYPE ABCDEFG.XYZ↲

;       MDS-800 I/O Drivers for CP/M 2.2
;       (four drive single density version)
;
;       Version 2.2 February, 1980
;
vers    equ     22      ;version 2.2
;
;       Copyright (c) 1980
;       Digital Research
;       Box 579, Pacific Grove
;       California, 93950
;
;
true    equ     0ffffh  ;value of "true"
false   equ     not true        ;"false"
        .
        .
        .
        .
```

Figure-9.7.4 リネーム後のファイル内容の確認

当然ですが，このように名前だけ変っていて，中身は同じです.

実習B　同一ディスク上にすでに存在しているファイル名にはリネームできません

Fig-9.7.5にそのサンプルランを示します．"PIP.COM"をすでに存在しているファイル名"ED.COM"にリネームしようと試みますが，"すでに在り"のメッセージが出力されるだけでリネームできません.

```
A>REN ED.COM=PIP.COM↲
FILE EXISTS
A>
```

Figure-9.7.5 既存ファイル名へのリネームはできない

実習C　ファイル・マッチ（*，?）は使用できません

REN コマンドは，新・旧どちらのファイルにも，ファイル・マッチ記号があると，エラー表示をして CP／M にもどります．そのサンプルランを Fig-9.7.6に示します．

```
A>REN BAD.*=ED.*↲
BAD.*=ED.*?

A>
```

Figure-9.7.6　ファイル・マッチは使えない

実習D　ログイン・ディスク以外のディスク上のファイル名の変更

Fig-9.7.8はドライブAにログインされた状態で，ドライブB上のファイル〝M80 . COM〟を〝12345. 999〟にリネームするサンプルランを示します． この場合， 右辺の〝B：〟は省略してもけっこうです．実行の前に，ドライブB上のファイルを DIR コマンドでタイプ・アウトして確認したものを Fig-9.7.7に示します．

```
A>DIR B:↲
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: LIB      COM : CREF80   COM : TESTPRO  BAS : FORLIB   REL
B: MBASIC   COM
A>
```

Figure-9.7.7　Bドライブのファイルの確認

```
A>REN B:12345.999=B:M80.COM↲
A>
```

Figure-9.7.8　Bドライブのファイルのリネーム

結果はどうか，もう一度 DIR コマンドで確認してみましょう．"12345.999" にリネームされています．これを Fig-9.7.9に示します．

```
A>DIR B:↲
B: BASLIB   REL : BASCOM   COM : 12345    999 : L80      COM
B: LIB      COM : CREF80   COM : TESTPRO  BAS : FORLIB   REL
B: MBASIC   COM
A>
```

Figure-9.7.9　リネーム後のファイルの確認

REN コマンド運用上のアドバイス

ここでも，前項の ERA コマンドで述べた「ERA コマンド運用上のアドバイス」が，そっくり当てはまります．それをもう一度参照されて運用して下さい．しかし，REN コマンドは ERA コマンドと違ってたとえ間違ったとしてもファイルが消えてなくなるわけではないので，"致命的" なミスは起こらないので安心です．

9.8 SAVE (セーブ)（メモリ内容のディスク・セーブ・コマンド）

—— SAVE コマンド ——

機能　アドレス100H からのメモリの内容を，任意のページ分(1ページ=256バイト)，任意のファイル名を付けて任意のディスクにセーブする．

SAVE コマンドを実習する前に，まずは，アドレス100H からのトランジェント・プログラム・エリア★と呼ばれるメモリ上に，何かデータを書き込んでおいた方がよいでしょう．そうすればディスクにセーブされた内容を後で確認することができます．

アドレス100H からの RAM にデータを書き込むには，ここではデバッガの DDT プログラムを使います．DDT プログラムについては後章で解説しますので，ここではサンプルランの通りにキーインするだけでけっこうです．

実習A　DDT プログラムで RAM にデータを書き込む

アドレス 100H から2FFHまでのメモリ2ページ（256バイト×2）に，

100H～1FFH=41H（ASCII の〝A〞）
200H～2FFH=58H（ASCII の〝X〞）

のコードを DDT プログラムを使って書き込むことにします．そのサンプルランを Fig-9.8.1に示します．

★ トランジェント・プログラム・エリア——トランジェント・プログラムがロードされるユーザー・エリア．100H から始まる

```
A>DDT↵
DDT VERS 2.2
-F100,1FF,41↵----100～1FFに "41" をフィル.
-F200,2FF,58↵----200～2FFに "58" をフィル.
-D100,2FF↵--------ダンプ
0100 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0110 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0120 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0130 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0140 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0150 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0160 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0170 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0180 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0190 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
01A0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
01B0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
01C0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
01D0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
01E0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
01F0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0200 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
0210 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
0220 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
0230 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
0240 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
0250 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
0260 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
0270 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
0280 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
0290 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
02A0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
02B0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
02C0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
02D0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
02E0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
02F0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 XXXXXXXXXXXXXXXX
-^C
A>
```

Figure-9.8.1 DDTプログラムでメモリにデータを書き込む

DDT については後章で解説しますが，ここで使った DDT の内部コマンドの〝F〟はフィル，〝D〟はダンプ，最後は ctrl-C で CP／M にもどっています．

実習B　ログイン・ディスクへのセーブ

アドレス100H から2ページ(1ページ=256バイト)分,512バイトのメモリの内容を,"TESTSAVE.AX" というファイル名で,同じログイン・ディスク上にセーブします.そのサンプルランを Fig-9.8.2に示します.数字の"2"はセーブしようとするページ数の指定です.

```
A>SAVE 2 TESTSAVE.AX↲
A>
```

Figure-9.8.2　メモリの内容を2ページ分セーブ

ディスクが2～3回アクセスされた後,CP/M にもどると実行終了です.

セーブされているかどうかの確認を,DIR コマンドと,後章で解説する STAT プログラムで行ったサンプルランを Fig-9.8.3～4に示します.

```
A>DIR TESTSAVE.AX↲
A: TESTSAVE AX
A>
```

Figure-9.8.3　"DIR"による確認

```
A>STAT TESTSAVE.AX↵

 Recs  Bytes  Ext Acc
    4     1k    1 R/W A:TESTSAVE.AX
Bytes Remaining On A: 103k

A>
```

Figure-9.8.4 STATプログラムによりその情報を見る

このように,現在のログイン・ディスクであるドライブA上に,1Kバイト長のファイル"TESTSAVE.AX"がセーブされていることが確認されます.512バイトしかセーブしなかったのに,1Kバイト長のファイルが出来ている理由は,CP/Mのファイル管理が,最少でも1Kバイトの単位で行われるからです.(大容量のハード・ディスクなどでは4Kとか8K単位になる)

では次に,実際に"TESTSAVE.AX"の内容をDUMPプログラム(後章で解説)を使って,ダンプしてみましょう.そのサンプルランをFig-9.8.5に示します.

```
A>DUMP TESTSAVE.AX↵

0000 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
0010 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
0020 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
0030 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
0040 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
0050 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
0060 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
0070 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
0080 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
0090 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
00A0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
00B0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
00C0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
00D0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
00E0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
00F0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
0100 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
0110 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
0120 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
0130 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
0140 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
0150 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
0160 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
0170 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
0180 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
0190 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
01A0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
01B0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
01C0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
01D0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
01E0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58
01F0 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58

A>
```

Figure-9.8.5 DUMPプログラムによりその内容を見る

このようにファイルの中身は,セーブしようとした RAM の内容と同一であることが確認されます.

実習C　SAVE コマンドによるファイルのコピー

DDT プログラムと，SAVE コマンドを使って，後述の PIP プログラムを使用せずにファイルをコピーすることができます．例としてダンプ・プログラムの〝DUMP.COM〞をコピーしてみましょう．手順はまず DDT で目的のファイルを100H からのトランジェント・プログラム・エリアへロードしてから，SAVE コマンドで，そのファイルの長さ分，ディスクへセーブします．

この DDT と SAVE コマンドを使ったファイルのコピー操作のサンプルランを Fig-9.8.6～7に示します．

```
A>DDT DUMP.COM↲
DDT VERS 2.2
NEXT  PC
0300 0100
-^C
A>
```

Figure-9.8.6 〝DUMP.COM〞をメモリへロード

```
A>SAVE 2 DUMPCOPY.COM↲
A>
```

Figure-9.8.7 〝DUMPCOPY.COM〞という名前を付けセーブする

詳しくは DDTの章で解説しますが，Fig-9.8.6の操作で，ファイル〝DUMP.COM〃が100H からの RAM へロードされます．ロードされたページ数は，DDT により表示された PC と NEXT の値で分ります．この例では100H から始まって，300H で終っています．つまり2ページ分ということを表わしています．もっと簡単にページ数を知る方法は，STAT プログラムで〝DUMP.COM〃を調べ，その Recs の項の数値を½にすれば，それがページ数になります．Fig-11.2.6を参照．

DDT の操作の後，SAVE コマンドを実行します．新しいファイル〝DUMPCOPY.COM〃が出来ているはずです．STAT プログラムでの確認のサンプルランを Fig-9.8.8に示します．

```
A>STAT DUMPCOPY.COM↲

 Recs  Bytes  Ext Acc
    4     1k    1 R/W A:DUMPCOPY.COM
Bytes Remaining On A: 102k

A>
```

Figure-9.8.8 〝DUMPCOPY.COM〃の情報を見る

このように新ファイルが出来ていることが確認されます．参考までに，ダンプ・プログラムである〝DUMP.COM〃のコピー・ファイル，〝DUMPCOPY.COM〃の内容を自分自身でダンプしてみましょう．そのサンプルランを Fig-9.8.9に示します．〝DUMP.COM〃の内容と比較しておいて下さい．当然ながら全く同一のはずです．

```
A>DUMPCOPY DUMPCOPY.COM↲

0000 21 00 00 39 22 15 02 31 57 02 CD C1 01 FE FF C2
0010 1B 01 11 F3 01 CD 9C 01 C3 51 01 3E 80 32 13 02
0020 21 00 00 E5 CD A2 01 E1 DA 51 01 47 7D E6 0F C2
0030 44 01 CD 72 01 CD 59 01 0F DA 51 01 7C CD 8F 01
0040 7D CD 8F 01 23 3E 20 CD 65 01 78 CD 8F 01 C3'23
0050 01 CD 72 01 2A 15 02 F9 C9 E5 D5 C5 0E 0B CD 05
0060 00 C1 D1 E1 C9 E5 D5 C5 0E 02 5F CD 05 00 C1 D1
0070 E1 C9 3E 0D CD 65 01 3E 0A CD 65 01 C9 E6 0F FE
        .
        .
        .
        .
```

Figure-9.8.9 〝DUMPCOPY.COM〃により自分自身をタイプアウト

実習D　ログイン・ディスク以外のドライブ上へのセーブ

今まで行ったSAVEのサンプルランは,ログイン・ディスク上にセーブする例ですが，これをログイン・ディスク以外のドライブ上にセーブしてみましょう．今までのコマンドと同様に，ファイル名の前にドライブ名〝x：〟を付ければよいのです.

Fig-9.8.10にドライブB上にセーブするサンプルランを示します.

```
A>SAVE 2 B:TESTSAVE.AX↲
A>
```

Figure-9.8.10 ログイン・ディスク以外へのセーブ

SAVE コマンド運用上のアドバイス

SAVEコマンドは,よくDDTプログラムと一緒に使われ,デバッグ中のパッチを行なったプログラムを一旦セーブする場合や，ファイルに少し変更を加えてコピーする場合など，主にマシン語開発時に使います.もしセーブするディスク上に同一名のファイルがすでに存在している場合，SAVEコマンドを実行すると，ディスク上にもとあった旧ファイルは削除されて，新しくセーブされたファイルと入れ替ってしまいますので，ファイル名を付ける時は十分注意して下さい.

SAVEコマンドは続けて何回でも実行することができます.メモリの内容を破壊することはありません.但しCP／Mのversion 1.4ではメモリの内容が変ってしまうので1回限りしか実行できませんので注意が必要です.

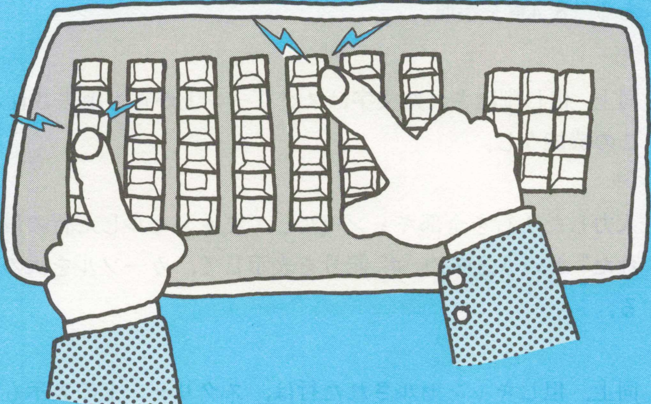
9.9 USER(ユーザー) (ユーザー・ナンバー指定コマンド)

—— USER コマンド ——

USER コマンドは，別段むつかしいコマンドではありません．1つのドライブに挿入されている1枚のディスケットを，仮想の16枚のディスケットに分割し，その16枚を1枚づつ単独に使用しているような効果を与えるコマンドです．

しかしこのコマンドは，本来マルチ・ユーザー・システムであるMP／Mのためのものであり，CP／Mでは大容量のハードディスクでも使用しない限り，使う意味がほとんどありません．よって本書では実習致しません．なお version 1.4にはこのコマンドはありません．

10章　キー入力時のライン エディッティング機能と 出力コントロール



コントロール・キーによるコマンド
各コントロール・キーの説明

10.1 コントロール・キーによるコマンド

コマンドをキー入力する際や，エディタ（ED）や DDT 内で文字列やサブ・コマンドをキー入力する際に，いくつかのライン編集機能が働きます．また，次々と表示されて行くスクリーン（コンソール）上の表示を一時ポーズ（フリーズ）状態にしたり，スクリーンに表示されるものを同時にプリンタ（リスト・デバイス）へ出力させることもできます．

これらの機能はキーボードからのコントロール・キャラクタによるコマンドで働き，次に示す合計10種類ほどのコマンドがあります．これらは Ctrl キーと共に用います．

10.2 各コントロール・キーの説明

1文字デリート	DEL/RUB 又は Back space	最後に入力した文字を1文字づつ削除する．削除された文字はマシンにより，スクリーン上から消去されるものと，再表示（エコーバック）されるものとがある．8.4章を参照．
	Ctrl-H	同上．但し削除された文字はスクリーン上からも消去される．version 1.4ではこの機能なし．
1ライン・キャンセル	Ctrl-U	入力した1行を全部キャンセルし，キャンセルした行の最後に〝キャンセルされた〟という意味の〝#〟記号を表示して，カーソルを次の行の先頭にセットする．
	Ctrl-X	同上．但しキャンセルされた行は，スクリーン上の表示も同時に消去される．カーソルは同じ行の先頭にもどる．version 1.4では ctrl-U と同じ機能．
ライン・ターミネート	Ctrl-M	Carriage Return と同じ．
	Ctrl-J	Line Feed と同じ．version 2.0 以上は Carriage Return の代用として使用可．

スクリーン・コントロール

Ctrl-S　スクリーン表示がスクロールしている時など，一時ポーズ（フリーズ）状態にする．再び ctrl-S 又は他のキーを入力するとキャンセルされる．

Ctrl-R　入力された1行を次の行にきれいに〝清書〞して再表示する．1文字削除でエコーバックされた見づらい行を確認する場合などに使う．Fig-10.2.1は最後の3文字を削除し，表示が見づらくなった行での実行例．

Ctrl-E　スクリーン上の表示のみに関する復帰・改行．このために入力されているコマンドが実行されることはない．

Ctrl-I　8文字ごとのタブ機能

Ctrl-P　スクリーンに表示されるものを〝LST：〞デバイス（通常はプリンタ）にも同時出力する．プリンタが動作可能な状態でないと，そこで hang-up(立ち往生）してしまうので注意すること.ctrl-P は入力のたびに ON-OFF をくり返すトグル動作になっている．

Ctrl-C　リブート（ウォーム・スタート，ウォーム・ブート）を起す．コマンド・ラインの最初に入力した時のみ有効．リブートに関しては8.5章を参照．

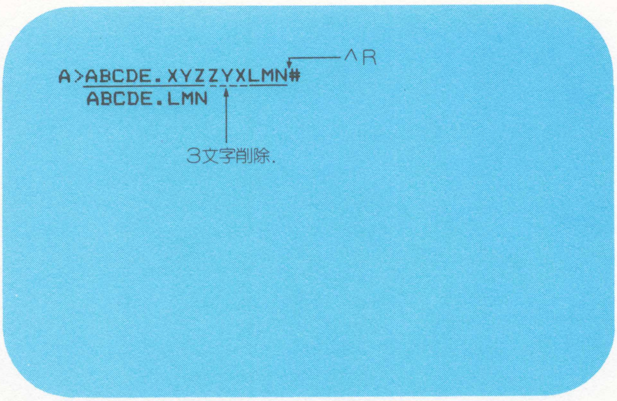
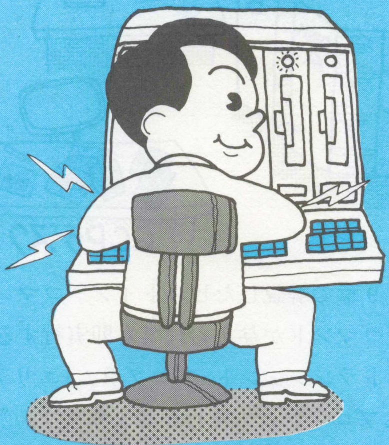


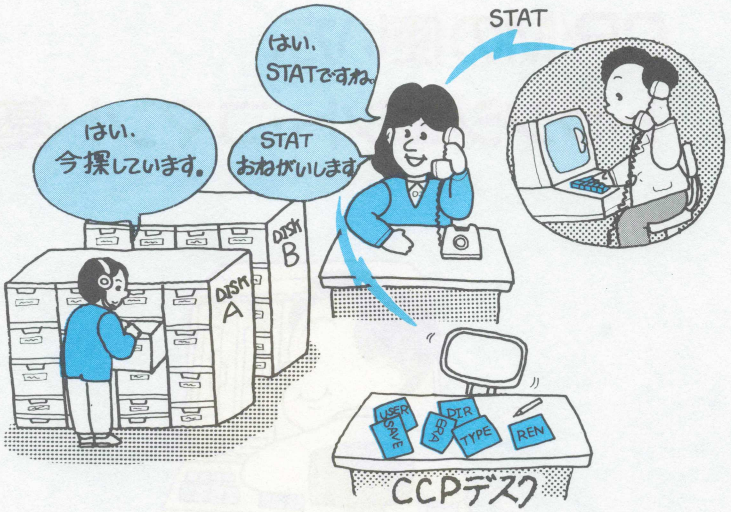
Figure-10.2.1 Ctrl-Rの実行例

11章　CP/Mの使い方，トランジェント・コマンド基礎実習



トランジェント・コマンド(プログラム)とは？
STAT(ファイルや周辺装置の設定および状況報告プログラム)
PIP(周辺装置間のデータ転送プログラム)
ED(テキスト・エディタ)
ASM(8080アセンブラ)
LOAD(HEXファイル⇨COMファイル変換プログラム)
DDT(8080デバッガ)
DUMP(ディスク・ファイルのダンプ・プログラム)
SUBMIT(バッチ処理・プログラム)
SYSGEN(CP/Mシステム生成プログラム)
MOVCPM(CP/Mシステム・リロケート・プログラム)

11.1 トランジェント・コマンド（プログラム）とは？



トランジェント・コマンドは，9章で解説したビルトイン・コマンドとは異り，コンピュータのメモリに常駐していません．よってコマンドが与えられても即実行することはできず，まずディスクから目的のプログラムを読み出し，トランジェント・プログラム・エリア（TPA と言う．アドレス 100H から始まる）にロードしてからプログラムの実行が始まります．〝トランジェント・コマンド〞という名称よりも〝トランジェント・プログラム〞と言った方が分り易いでしょう．本書では以後，トランジェント・プログラムと呼ぶことにします．

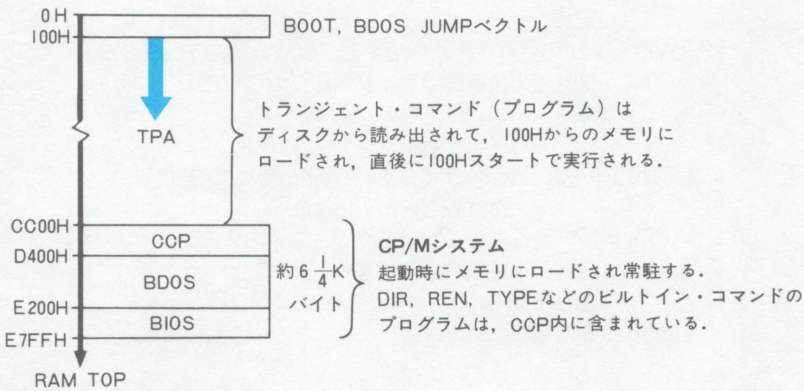


Figure-11.1.1 58K CP／Mが起動した時のメモリ・マップ

11.2 STAT（スタット） （ファイルや周辺装置の設定および状況報告プログラム）

——STATistical information program——

機能　ディスク上のファイルの状態，ディスク・ドライブの諸元，USER 番号のレポートなどを行う．またファイルのアトリビュート（属性）のレポートおよび指定，周辺装置の割当てと，その状態レポートなども行う．

実習A　ディスクの未使用エリアの容量を調べる

CP／M が起動して,まだ一度もドライブA以外のドライブにアクセスしていない場合は,A以外のドライブに関してのディレクトリをメモリ内に読み込んでいないため，ドライブAに関してのみレポートします．そのサンプルランを Fig-11.2.1に示します．

```
A>STAT↲
A: R/W, Space: 106k


A>
```

Figure-11.2.1 ディスクの未使用エリアを調べる

一度でも他のドライブをアクセスした場合は，そのドライブの未使用エリアの容量も一緒にレポートします．

Fig-11.2.2で，わざとドライブBにログインした後，再びAに戻し，STAT コマンドを実行します．それを Fig-11.2.3に示します．

```
A>B:↵
B>A:↵
A>
```

Figure-11.2.2 1度Bドライブにログインする

```
A>STAT↵
A: R/W, Space: 106k
B: R/W, Space: 69k

A>
```

Figure-11.2.3 再度未使用エリアを調べる

このように，ディスクAはリード／ライト可能なスペースが 106K バイト，ディスクBは 69K バイト残っていることがレポートされています.

実習B　ファイルの長さ，レコード長，アトリビュートの状態などを調べる

まず現在のログイン・ディスクであるドライブA上のファイル〝BIOS.ASM〞についてのサンプルランを Fig-11.2.4に示します.

```
A>STAT BIOS.ASM↲

 Recs  Bytes  Ext Acc
   96    12k    1 R/W A:BIOS.ASM
Bytes Remaining On A: 106k

A>
```

Figure-11.2.4 "BIOS.ASM" の情報を見る

この例では，ファイル"BIOS.ASM"は，ファイル長(Bytes)が 12K バイトで，レコード長(Recs)が96，このファイルによって占められるロジカル・エクステント数（Ext：1エクステント＝16K バイト）は1，ファイルのアトリビュート(Acc)はリード／ライト可能ファイルを表し，このディスケットの残り容量は 106K バイトであることがレポートされています．

Fig-11.2.5は，同様にドライブB上のファイル"MBASIC.COM"について調べるサンプルランです．他のコマンドでも実習したように，ファイル名の前にドライブ名"x："を付けて実行します．

```
A>STAT B:MBASIC.COM↲

 Recs  Bytes  Ext Acc
  188    24k    2 R/W B:MBASIC.COM
Bytes Remaining On B: 69k

A>
```

Figure-11.2.5 Bドライブの"MBASIC.COM"の情報を見る

このようにマイクロソフト社の MBASIC (BASIC-80) はファイル長が 24K バイトであることが分ります．

STAT コマンド関連用語の説明

Recs ：レコード・ブロック数．1レコードは128バイト．CP／M では，ロジカルな1セクタは，すべてのディスクで128バイトである．

Bytes：ファイルの実際の全容量キロバイト数．

Ext ：CP／M のファイル管理上の16 K バイト単位のロジカルなエクステント数．

Acc ：ファイル・アトリビュート．〝属性〟などと，あまりよく分らない訳語が使われている．「このファイルは何々の性質を持つ種類に〝属〟している」という意味であり，ファイルの性質上の分類の名称である．ファイル名の〝エクステンション〟と混同しないように．

$R／W……リード／ライト可能な性質．

$R／O……書き込み・削除禁止の性質．

など．

実習C ある条件にマッチしたファイルの状態を調べる

ファイルの状態を調べるのに，ファイル・マッチを使うことができます．Fig-11.2.6にログイン・ディスク上のすべての〝COM〟エクステンションを持つファイルをレポートするサンプルランを示します．

```
A>STAT *.COM
 Recs  Bytes  Ext Acc
   64     8k    1 R/W A:ASM.COM
   38     5k    1 R/W A:DDT.COM
    4     1k    1 R/W A:DUMP.COM
   52     7k    1 R/W A:ED.COM
   14     2k    1 R/W A:LOAD.COM
   76    10k    1 R/W A:MOVCPM.COM
   58     8k    1 R/W A:PIP.COM
   68     9k    1 R/W A:PTCPM48.COM
   41     6k    1 R/W A:STAT.COM
   10     2k    1 R/W A:SUBMIT.COM
    8     1k    1 R/W A:SYSGEN.COM
    6     1k    1 R/W A:XSUB.COM
Bytes Remaining On A: 106k

A>
```

Figure-11.2.6 ファイル・マッチを使って情報を見る

実習D　任意のファイルにアトリビュートを設定する

例として，ログイン・ディスク上の〝DUMP.COM〟を，書き込みや，削除が出来ない R／O（リード・オンリー）アトリビュートを持つファイルに変更します．現在は R／W（リード／ライト）アトリビュートを持っていることが Fig-11.2.6でわかります．

そのサンプルランを Fig-11.2.7に示します．

```
A>STAT DUMP.COM $R/O↲

DUMP.COM set to R/O
A>
```

Figure-11.2.7 R／Oアトリビュートを設定する

これで R／O アトリビュートが付きました．参考までにこのファイルの状態を見てみましょう．Fig-11.2.8に示します．

```
A>STAT DUMP.COM↲

 Recs  Bytes  Ext Acc
    4     1k    1 R/O A:DUMP.COM
Bytes Remaining On A: 106k

A>
```

Figure-11.2.8 アトリビュートを確認する

このように Acc の欄は R/W から R/O に変っています.

では実際に消去不可能かどうか ERA コマンドを実行してみましょう.そのサンプルランを Fig-11.2.9に示します.

```
A>ERA DUMP.COM
Bdos Err On A: File R/O ^C

A>
```

Figure-11.2.9 書き込み禁止ファイルを削除してみる

このように実際に "DUMP.COM" は削除できないことが分ります. これらのアトリビュートは,それぞれのファイルについて,ディスクに書き込まれますので,リブートや,再起動を行っても失なわれません.書き込みや,削除したい場合は,あらためて R/W アトリビュートを付ければ可能になります.

実習E コマンド・メニューと IO バイトのアサインの状況をディスプレイする

```
A>STAT VAL:

Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
User Status  : USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
A>
```

Figure-11.2.10 コマンド・メニューを表示する

このように STAT コマンドのメニューが表示されます．このメニューの指示通りにコマンドを実行してみましょう．3行目の Disk Status を行ってみます．表示によると，

STAT DSK：　あるいは

STAT d：DSK

とやればよいわけです．そのサンプルランを Fig-11.2.11に示します．

```
A>STAT DSK:↲

    A: Drive Characteristics
 1944: 128 Byte Record Capacity
  243: Kilobyte Drive  Capacity
   64: 32  Byte Directory Entries
   64: Checked  Directory Entries
  128: Records/ Extent
    8: Records/ Block
   26: Sectors/ Track
    2: Reserved Tracks

A>
```

Figure-11.2.11 ディスクの諸元を表示する

このようにログイン・ディスク（現在はA）のドライブの諸元が表示されます．ディレクトリ・エリアを含む全体で243 K バイトの容量があり，64個のファイルを収容可能で，CP／M のシステムは2トラックに記録されていることなどが示されています．

IO バイトのアサインについて

Fig-11.2.10の Iobyte Assign の欄は，〝=〟の左側をロジカル･デバイスと呼び，右側をフィジカル･デバイスと呼びます．CP／M が扱う周辺装置は，〝CON〟コンソール，〝RDR〟リーダー，〝PUN〟パンチャー，〝LST〟リスト装置，この4つのロジカル･デバイスであり，それぞれの装置名は付いていても，あくまでロジカル的な意味であり，〝=〟の右側に示されているフィジカル装置のいずれかを任意に割り当てて，使用するものです．

CP／M の起動時は，初期設定されているデバイスに割当てられていますが，その後，STAT コマンドにより IO バイトと呼ばれるアドレス 0003H の1バイトを操作することができ，〝=〟の右側の4つのデバイスの中から，1つを自由に割り当てることができるのです．

これらについての詳しい解説と実習は本シリーズの続巻で行います．

11.3 PIP（ピップ） (周辺装置間のデータ転送プログラム)

—— Peripheral Interchange Program ——

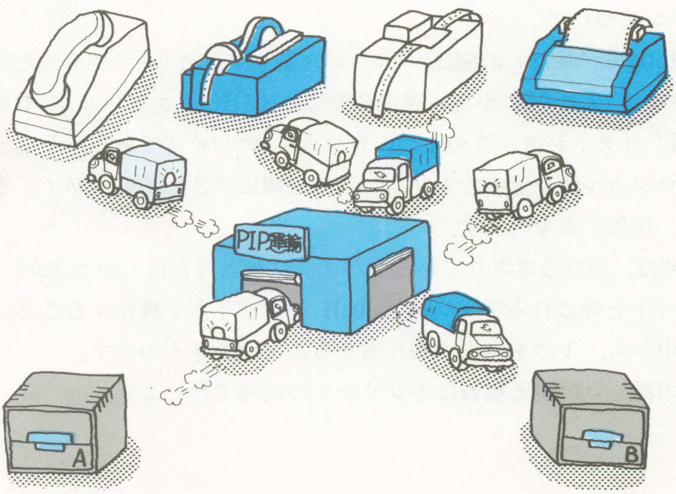
機能　PIPは，周辺装置間のデータ転送プログラムであり，PIP コマンドにさらに，〝PIP パラメータ〟と呼ぶオプション・コマンドを付けることにより，データ転送時にいろいろな処理を行うことが可能です．

PIP コマンドも実に様々の使い方が出来るコマンドですが，ここではその基本的なものを実習します．

PIP コマンドは，２つの実行方法があります．

1. メイン・コマンド PIP の後に続けて，内容を表わすコマンド・ラインを１行で記述し，PIP プログラムのロードと同時にそのコマンド・ラインを実行する方法．この場合，実行が終了すれば CP／M に戻る．１つだけの PIP 処理の場合用いるとよい．
2. メイン・コマンドの PIP プログラムのみ単独でまず起動しておき，PIPが起動したことを示すプロンプト〝*〟が出力された後に内容を表わすコマンド・ラインをキーインし，それを実行する方法．この場合，実行が終了すれば再び PIP プロンプト〝*〟にもどるので，次に新しい別の内容のコマンド・ラインをキーインし，次々とそれらのコマンドを実行することができる．

まず最初に，PIP プログラムによるディスクからディスクへのファイルのコピーを，この２つの方法で実習してみましょう．



実習A　ドライブAに挿入されているディスク上のファイルを，ドライブBにコピーする．

（書き込みを伴うコマンドやプログラムの実行の前に，もしディスケットなどを交換したりすると，自動的にそのドライブは書込み禁止になりますので，その場合は必ず ctrl-C でリブートをするか，コールド・スタートをしてから始めて下さい．）

まず1の方法でのサンプルランを Fig-11.3.1に示します．PIP コマンドの形式はすべて，〝=〟の左辺が受け取り側，右辺が送り出し側になります．〝B：〟の後にファイル名が省略されている場合は，送り出しのファイル名と同一のものが付けられます．送り側のファイル名〝STAT.COM〟の前に本来付けるべきドライブ名（この場合 A：）は，それがログイン・ディスクである場合にはこのように省略できます．

```
A>PIP B:=STAT.COM↲

A>
```

Figure-11.3.1 Bドライブに〝STAT.COM〟をコピーする

次に2の方法でのサンプルランを Fig-11.3.2に示します．

まず PIP コマンドを単独で実行・起動します．PIP が起動すると，PIP のプロンプト〝*〟が出力されるので，それに続いて先程の1.の場合と同様のコマンド・ラインをキーインし実行します．実行が終ると CP／M には戻らずに，再び PIP のプロンプトに戻りますので，続けて新しい PIP 操作を実行することができます．このサンプルランではリターンをキーインして PIP を終了させ CP／M に戻っています．

```
A>PIP↵
*B:=STAT.COM↵
*↵

A>
```

Figure-11.3.2 PIPを起動し〝STAT. COM〟をコピーする

以上2つのサンプルランは，どちらもドライブBのディスク上にファイル，〝STAT．COM〟がコピーされます．それを DIR で確認したものを Fig-11.3.3に示します．

```
A>DIR B:STAT.COM↵
B: STAT     COM
A>
```

Figure-11.3.3 コピー後のファイルの確認

またこれらのディスク間のファイル・コピーは，受け取る側のファイル名を自由に変えることが可能です．ファイル名を特に指定しない場合は，送り出し側のファイル名と同一のものが自動的に付けられます．

次は，この受け取り側のファイル名を指定してファイルをコピーするサンプルランを Fig-11.3.4に示します．

```
A>PIP B:STATPIP.COM=STAT.COM↲

A>
```

Figure-11.3.4 ファイル名を替えてコピーする

これでドライブA上のファイル〝STAT.COM〟が，ドライブB上に，〝STATPIP.COM〟としてコピーされました．DIR コマンドで確認したものを Fig-11.3.5に示します．

```
A>DIR B:STATPIP.COM↲
B: STATPIP  COM
A>
```

Figure-11.3.5 コピー後のファイルの確認

ではここで，ドライブB上に出来たSTAT プログラムである，〝STAT.COM〟のコピー・ファイル，〝STATPIP.COM〟を実行してみましょう．ログイン・ディスクをBにチェンジしてから行うことにします．そのサンプルランを Fig-11.3.6～7に示します．

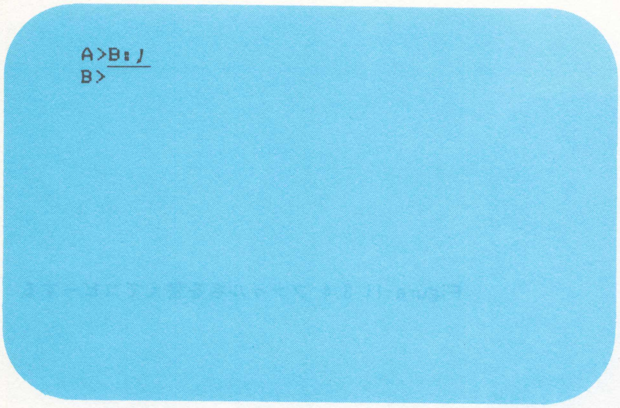


Figure-11.3.6 Bドライブに移行



Figure-11.3.7 コピーした〝STATPIP.COM〟を実行

当然ですがこのように STAT コマンドであることが分ります．その前に行った同一ファイル名でのコピーにより，ドライブB上にセーブされた〝STAT.COM〟を実行しても Fig-11.3.7と同様の結果が得られます．

実習B　任意のディスク上のファイルを任意のディスクにコピーする

ログイン・ディスクはBで,ドライブA上のPIP プログラムを使って,ドライブA上のファイル"DISKDEF.LIB" をドライブB上にコピーします．サンプルランを Fig-11.3.8に示します．

```
B>A:PIP B:=A:DISKDEF.LIB↲
B>
```

Figure-11.3.8 ドライブA上の"PIP"を起動しコピーする

では DIR コマンドでドライブB上にコピーされているか見てみましょう．Fig-11.3.9に示します．

```
B>DIR DISKDEF.LIB↲
B: DISKDEF  LIB
B>
```

Figure-11.3.9 コピー後のファイルの確認

このようにコピーされています．

またオリジナル(ソース)ファイルのディスケットを交換しながら PIP を実行することもできます．

最初に述べたように PIP の実行方法は２つあるので，Fig-11.3.8は次の Fig-11.3.10のように書いても結果は同じです．

```
B>A:PIP↲
*B:=A:DISKDEF.LIB↲
*     .
*     .   続けて次々とPIPコマンドを実行できる．
*     .
*

B>
```

Figure-11.3.10 〝PIP〞を起動しておきコピーする

この２つのコマンドの書き方は，結果は同じですが，Fig-11.3.10のように，最初に PIP プログラムのみを起動して，PIP のプロンプト〝＊〞に従ってコマンド・ラインを実行する方法の場合，

　　オリジナル・ファイル（ソース・ファイル）側のディスケットを交換できる

という重要な利点があります．

Fig-11.3.10の例の場合は，ディスケットが別々のいくつかのファイルをドライブＡ上で交換しながら PIP のプロンプト〝＊〞に従い，次々とドライブＢ上にコピーすることが可能であるわけです．

実習Ｃ　ドライブＢにドライブＡ上のすべての〝COM〞ファイルをコピーする

PIP はファイル・マッチが使えます．

まずドライブＢのディスケットの空エリアが少ないかも知れませんので念のために，ディスクＢのすべてのファイルを消去しておきましょう．この作業は Fig-11.3.11のように行うことは ERA コマンドの項で実習しました．

```
B>ERA *.*
ALL (Y/N)?Y
B>A:
A>
```

Figure-11.3.11 あらかじめすべてのファイルを削除しておく

これでディスクBは，〝空〟になりました．同時にログイン・ディスクをAにチェンジしています(実習を分り易くするため)．ドライブA上のすべての〝COM〟エクステンションを持つファイルを，ドライブB上にコピーするサンプルランを Fig-11.3.12に示します．

```
A>PIP B:=*.COM

COPYING -
MOVCPM.COM
PIP.COM
SUBMIT.COM
XSUB.COM
ED.COM
ASM.COM
DDT.COM
LOAD.COM
STAT.COM
SYSGEN.COM
DUMP.COM
PTCPM48.COM

A>
```

Figure-11.3.12 〝COM〟エクステンションを持つファイルをすべてコピーする

このように〝COM〟ファイルだけが次々とコピーされて行き，全部が終ると，CP／M に戻っています．

これと同様に，すべてのファイルをコピーするには，〝＊．＊〟を使えばよいわけです．これは8.7章で実習しましたので参照して下さい．

実習D　オプション・パラメータを使ってのコピー

ドライブB上のファイル〝MBASIC.COM〟を現在のログイン・ディスクであるAにコピーします.コピーに際しては，リード・アフタ・ライトの厳重なチェックを行わせ，コピーされた内容を保障します.このサンプルランを Fig-11.3.13に示します.

```
A>PIP A:=B:MBASIC.COM[V]↲

A>
```

Figure-11.3.13 ベリファイ・オプションを付けてのコピー

コマンド・ラインの最後の〝[V]〟がベリファイ・オプション・パラメータと言って，ディスクへの書き込み後，即読み出して，書き込むべきデータと比較チェックを行わせるオプション・コマンドです.

Aにコピーされた〝MBASIC.COM〟を DIR で確認したものを Fig-11.3.14に示します.

```
A>DIR MBASIC.COM↲
A: MBASIC   COM
A>
```

Figure-11.3.14 実行後の確認

オプション・パラメータについてはこの後，２～３実習しますが，簡単な例として，［Ｅ］パラメータがあります．これは周辺装置間で転送されているデータを同時にコンソールにも表示（エコーバック）するものです．よって今まで実習したディスク間のファイル・コピーに於ても，ファイルがアスキー・ファイルであれば，この〝［Ｅ］〞を付けることにより転送されている内容が，直接スクリーン上で見ることができるわけです．その一つのサンプルランを Fig-11.3.15に示します．

```
A>PIP B:=CBIOS.ASM[E]

;       Skeletal CBIOS for first level of CP/M 2.0 alteration
;
msize   equ     20      ;cp/m version memory size in kilobytes
;
;       "bias" is address offset from 3400H for memory systems
;       than 16K (referred to as "b" throughout the text).
;
bias    equ     (msize-20)*1024
ccp     equ     3400H+bias      ;base of ccp
bdos    equ     ccp+806h        ;base of bdos
bios    equ     ccp+1600h       ;base of bios
cdisk   equ     0004H   ;current disk number 0=A,...,15=P
iobyte  equ     0003h   ;intel i/o byte
;
        org     bios    ;origin of this program
nsects  equ     ($-ccp)/128     ;warm start sector count
     .
     .
     .
     .

A>
```

Figure-11.3.15 エコー・バック・オプションを付けてのコピー

これは，ファイル〝CBIOS．ASM〞をディスクＢにコピーするものですが，同時に転送内容がスクリーン上に表示されます．

では次に，周辺装置にファイルを転送する方法について実習してみましょう．

実習E　コンソール・デバイスに任意のファイルを出力する

まず周辺装置として，コンソール（ロジカル・デバイス名 CON：）ヘディスク上のアスキー・ファイルを出力する実習です．

現在のログイン・ディスク上のファイル〝PTBIOS48.ASM〞を CON：デバイスに出力するサンプルランを Fig-11.3.16に示します．普通，CON：デバイスは CRT スクリーンですので，スクリーンにデータが表示されて行きます．

```
A>PIP CON:=PTBIOS48.ASM↲

; CP/M BASIC INPUT/OUTPUT OPERATING SYSTEM (BIOS)
; TARBELL ELECTRONICS
; 2.2 VERSION OF 2-19-80
; CHANGED SIGN-ON FOR CPM 2.2 2-19-80
;
;===== CUSTOMARISED BY YASUHARU MURASE =====
;=====              8-6-80              =====
;
; THIS MODULE CONTAINS ALL THE INPUT/OUTPUT
; ROUTINES FOR THE CP/M SYSTEM, INCLUDING
; THE DISK ROUTINES.
;
; THIS SECTION DEFINES THE I/O PORTS AND
; STATUS BITS.  BY SETTING THE PROPER VALUES
; FOR THE EQU STATEMENTS, THE I/O MAY BE
; AUTOMATICALLY RECONFIGURED TO FIT MOST
; SITUATIONS.  THE TRUE AND FALSE ONES
; CONTROL CONDITIONAL ASSEMBLIES OF DIFFERENT
; SECTIONS OF I/O ROUTINES TO FIT DIFFERENT
; INTERFACE REQUIREMENTS.
        .
        .
        .
        .

A>
```

Figure-11.3.16 コンソール・デバイスに出力

この例では，TYPE コマンドと同じで，何もトランジェント・プログラムの PIP など使うことはないわけですが，PIP を使うと TYPE コマンドでは不可能な処理をオプション・パラメータにより行うことができるのです．

次のサンプルラン Fig-11.3.17をご覧下さい．

```
A>PIP CON:=PTBIOS48.ASM[ND30]↲

     1: ; CP/M BASIC INPUT/OUT
     2: ; TARBELL ELECTRONICS
     3: ; 2.2 VERSION OF 2-19-
     4: ; CHANGED SIGN-ON FOR
     5: ;
     6: ;===== CUSTOMARISED BY
     7: ;=====            8-6-8
     8: ;
     9: ; THIS MODULE CONTAINS
    10: ; ROUTINES FOR THE CP/
    11: ; THE DISK ROUTINES.
    12: ;
    13: ; THIS SECTION DEFINES
    14: ; STATUS BITS.  BY SET
    15: ; FOR THE EQU STATEMEN
    16: ; AUTOMATICALLY RECONF
    17: ; SITUATIONS.  THE TRU
    18: ; CONTROL CONDITIONAL
    19: ; SECTIONS OF I/O ROUT
    20: ; INTERFACE REQUIREMEN
      .
      .
      .
      .

A>
```

Figure-11.3.17 複数のオプション・パラメータをつけて転送

これは，先程の Fig-11.3.16のコマンド・ラインの最後に，オプション・パラメータの〝[ND30]〞を付けただけですが，出力はこのようになりました．〝N〞によりライン NO. が付き，〝D30〞により1行の字数が30字に制限されています．

このようにオプション・パラメータは，複数個を［　］の中にまとめて記述することができ，各種の出力の制御が同時に可能となっています．

実習F　プリンタ（PRN：装置）にページ割り付けしたリストを出力する

もう一つ，よく使われるサンプルランを Fig-11.3.18に示します．

これは周辺装置に〝PRN：〞（プリンタ）を指定したもので，この〝PRN：〞は PIP 上の特別なロジカル装置であり，この装置への出力は，ライン No. が付き，TAB の動作が行われ，60行毎にペー

ジ割付けが行われます．各種のリストを取る時などに，よくこの PIP コマンドが使われます．

```
A>PIP PRN:=B:BASIC5.ASM ↲
```

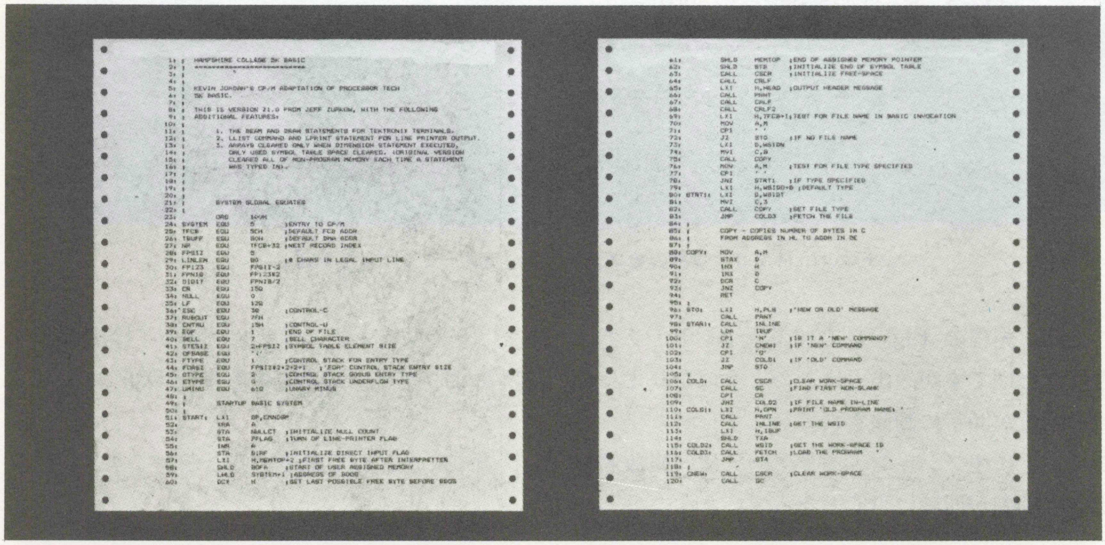


Figure-11.3.18 プリンタにページ割り付けしたリストを出力する

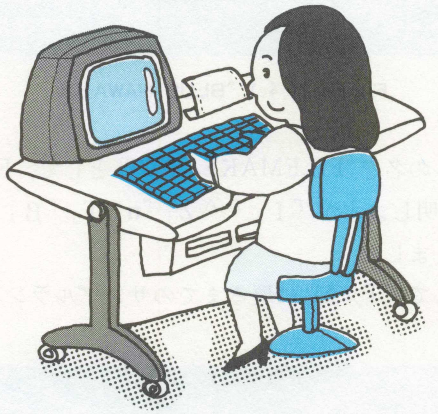
このように60行毎のページ送りが行われています．

PIP プログラムは，ここで２～３の実習をしたように，周辺装置の指定と，オプション・パラメータの指定により，非常に多機能な周辺装置間のデータ転送処理が行えます．周辺装置は，STAT プログラムなどで自由にアサイン可能であり，オプション・パラメータは20種ほど用意されています．これらの使用は，かなり専門的にもなり，詳細は本書の続巻で解説されます．

11.4 ED (テキスト・エディタ)
エディタ

—— EDitor ——

機能　新しいテキスト・ファイルの作成，既存テキスト・ファイルの編集.



我々とコンピュータとの接点になるのがこのエディタです．エディタによって作成されたプログラムが我々の意志をコンピュータに伝え，様々のことを実行させるわけです．CP／M のエディタは，スクリーン・エディタの機能こそありませんが，大型コンピュータのエディタと比較しても，その使い易さはひけをとらないと言われている名エディタです．ED 内のサブ·コマンドは種類が多く，慣れるまでは苦労しますが，一日も早くこの強力な名エディタを使いこなせるようになって下さい．又，ED はワード・プロセッサとしても使え，言わばその元祖とも言うべきものです．

本書では，すべてのサブ・コマンドについては解説を行っていませんが，日常よく使われる主なものについては，実例を示して解説しています．

実習A　新しいテキスト・ファイルの作成

新しいテキスト・ファイルを作成し，ディスクにセーブします．本来なら，アセンブラや高級言語のソース·プログラムを作ることが多いのですが，ここでは筆者の若かりし頃，1962年の映画「BLUE HAWAII」で Elvis が唄った歌の一節をファイルにしてみましょう．その文章を次の Fig-11.4.1に示します．

```
        CAN'T HELP FALLING IN LOVE

Wise men say only fools rush in
But I can't help falling love with you
Shall I stay would it be a sin
If I can't help falling love with you

             . . . . . . song by E.PRESLEY
```

Figure-11.4.1 〝BLUE HAWAII〞

では今から作成するファイルの名を〝FILEMAKE.TXT〞として，ED を起動します．ここで使う ED 内サブ・コマンドは後で説明しますが「I」「∧Z」「n：T」「B」「#T」「E」です．実際には小文字でこれらのコマンドを与えました．

この作業の開始から，終了して CP／M に戻るまでのサンプルランを Fig-11.4.2に示します．

```
A>ED FILEMAKE.TXT↲ -------ファイル名を指定してEDを起動する.

NEW FILE-----(新しいファイルであることの応答)
     : *i↲ -------- Iコマンドでインサート・モードに入る.小文字に注意.
    1:          CAN'T HELP FALLING IN LOVE↲   ┐
    2:  ↲                                     │
    3:  Wise men say only fools rush in↲      │
    4:  But I can't help falling live with you↲ │ 内容をタイプインして行く.
    5:  Shall I stay would it be a sin↲       │ 行の終りはキャリッジ・リターンで.
    6:  If I can't help falling love with you↲ │
    7:                                        │
    8:              . . . . . . song by E.PRESLEY↲ ┘
    9:  ∧Z------Ctrl-Z でインサート・モードを終る.   (ラインNo.4にミス・タイプ「live」
     : *4:t↲------ラインNo.4にCPを移動して，その行をタイプする.  があったので訂正作業をする.)
    4:  But I can't help falling live with you
    4: *slive^Zlove^Z0tt↲ ----- Sコマンドでliveをloveに置き替えてタイプする.
    4:  But I can't help falling love with you
    4: *b↲------CPを行の先頭へ移動.
    1: *#t↲-----#記号ですべての行をタイプする.
    1:          CAN'T HELP FALLING IN LOVE    ┐
    2:                                        │
    3:  Wise men say only fools rush in       │
    4:  But I can't help falling love with you │ (確認のためのタイプ・アウト)
    5:  Shall I stay would it be a sin        │
    6:  If I can't help falling love with you │
    7:                                        │
    8:              . . . . . . song by E.PRESLEY ┘
    1: *e↲ ------ディスクにセーブし，EDを終了する

A>
```

Figure-11.4.2 EDを使って〝FILEMAKE.TXT〞を作る

1字ミス・タイプがあったので訂正作業をし，目的のファイル〝FILMAKE．TXT〞が出来上ったようです．TYPE コマンドでその内容を確認してみましょう．サンプルランを Fig-11.4.3に示します．

```
A>TYPE FILEMAKE.TXT↲

      CAN'T HELP FALLING IN LOVE

Wise men say only fools rush in
But I can't help falling love with you
Shall I stay would it be a sin
If I can't help falling love with you

           . . . . . . song by E.PRESLEY

A>
```

Figure-11.4.3 〝FILEMAKE．TXT〞を見る

このように Fig-11.4.1に示したソース・テキストが，ディスクにセーブされていることが確認されました．

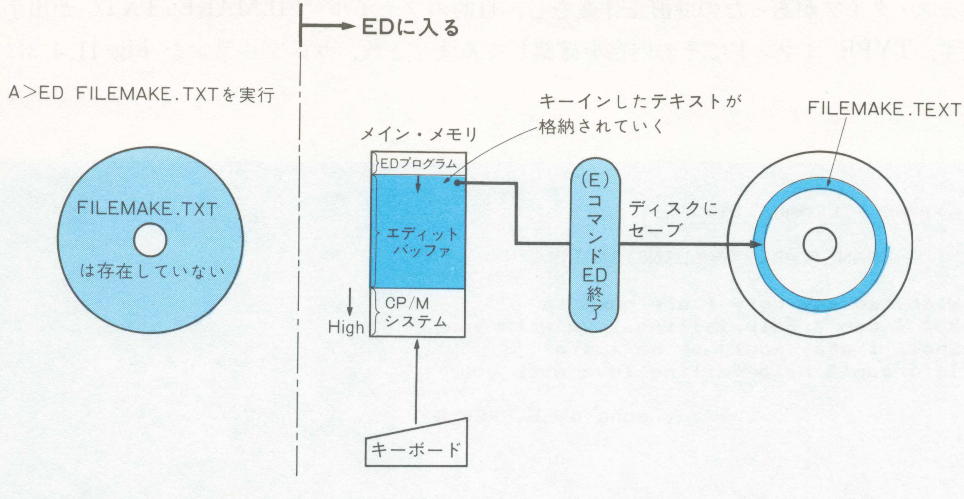
ここで，コンピュータの内部とディスクに，どんなことが起ったのか，その流れを説明しましょう．そのメカニズムを知ることは今すぐ必要というわけではありませんが，一応心得ておいた方がよいでしょう．

Fig-11.4.4をご覧下さい．新しいファイルを作るのですから，最初はそのファイル（この例では FILEMAKE．TXT）はディスク上に存在していません．

A＞ED FILEMAKE．TXT

を実行することにより，メイン・メモリのアドレス100Hから始まるトランジェント・プログラム・エリア（TPA）に，ED プログラムがディスクからロードされ，ED が起動します．ED プログラムがロードされた後方のメモリは，〝エディット・バッファ〞と呼ばれるエディッティング作業を行うためのエリアとして使用されます．（I）コマンドによりインサート・モードに入った後は，文字列をキーインし，最後にリターンする毎に1ラインずつ，すべてこのエディット・バッファに格納されて行きます．

エディッティング作業が終ると，（E）コマンドを使って，ED プログラムを終了させます．（E）コマンドが実行されると，エディット・バッファ内に完成している今作成したテキストは，ディスク上にセーブされ，ED プログラムを起動する時に指定したファイル名（ここでは FILMAKE．TXT）が



（注．実際のディスク上のファイルは，必ずしも図のように連続していません）

Figure-11.4.4 新しいファイルを作成する場合の流れ

付けられます．ディスクへのセーブが終ると，ED プログラムは完全に終了し，CP／M に戻ります．
以上が ED プログラムで新ファイルを作成する時のメカニズムの概略です．

実習B　既存テキスト・ファイルの編集

次は，ED 内のよく使われるサブ・コマンドを実習する為に，すでにディスク上に存在するファイル，"FIB. PLI"をエディットしてみましょう．このテキスト・ファイルは，CP／M 上で実行する Digital Research 社の PL／I-80コンパイラのサンプル・ソース・プログラムであり，フィボナッチ数列を求めるプログラムです．

TYPE コマンドでタイプアウトしたものを Fig-11.4.5に示します．

```
A>TYPE FIB.PLI↲

fibonacci:
    proc options(main);
    dcl i fixed;
    do i = 0 to 100;
    put list(fib(i));
    end;

    fib:
        proc(n) returns(fixed) recursive;
        dcl n fixed;
        if n = 0 then
            return(1);
        if n = 1 then
            return(1);
        return(fib(n-1) + fib(n-2));
        end fib;
    end fibonacci;

A>
```

Figure-11.4.5 編集前の〝FIB.PLI〞ファイル

エディットの内容については特に目的はなく，ただ各コマンドを実習するために行うだけです．

現在ログインされているドライブA上に，ED プログラムと，FIB.PLI が在るものとして，実習を始めましょう．

まずテキスト・ファイルの FIB.PLI に対し，ED を起動します．そのサンプルランを Fig-11.4.6に示します．

```
A>ED FIB.PLI↲
    : *
```

ファイル〝FIB.PLI〞に対してEDを起動．
EDのプロンプト〝*〞が出力される．

Figure-11.4.6〝FIB.PLI〞に対しEDを起動

このように ED が起動すると〝:〞に続いて，ED のプロンプトである〝*〞が表示され，ED のサブ・コマンドのキー入力待ちになります．version 1.4の CP/M では，ED が起動した状態ではライン No. が表示されません．ライン No. を表示させるためにはコマンド〝V〞が用意されているので，V コマンドをキーインすることにより，ライン No. が表示されるようになります．同時に〝:〞も表示されます．CP/M 1.4において，ライン No. を表示させるためのVコマンドのサンプルランを Fig-11.4.7に示します．

```
A>ED FIB.PLI↲
*V↲
     : *
```

Figure-11.4.7 Vコマンドを実行

これで version 2.2と同様の表示になりました．

さて，ED が起動して，プロンプト〝*〞が表示されたら，テキスト・ファイルである FIB.PLI をディスクから，メイン・メモリ内の〝エディット・バッファ〞にロードします．エディット・バッファは，Fig-11.4.4に図示されていますが，このメモリ上のエリアで，いろいろなエディッティング作業が行われるのです．このサンプルランを Fig-11.4.8に示します．

```
A>ED FIB.PLI↲
     : *#A↲
    1: *
```

(A) コマンドと〝#〞でテキストのすべてのラインをエディット・バッファにロードする．

Figure-11.4.8 Aコマンドによりすべてのテキストをバッファへロード

この〝#〟記号は，使用可能な最大のライン数である数字の65535を意味し，〝全部〟

#A　=　65535A

と同じことを表わします．エディット・バッファにテキストがロードされると，ED プロンプト〝*〟の前には，現在，キャラクタ・ポインタ（CP と略称，詳細は後述）があるライン No. が表示されます．Fig-14.4.8の例では，ラインをアペンド（Aコマンド）した直後なので，CP はライン No.1 にあり，1が表示されています．

エディット・バッファにロードされたテキストは，Tコマンドによって表示され，Fig-11.4.9のサンプルランでは，〝#T〟により，全部のラインがタイプアウトされています．

```
A>ED FIB.PLI↲
    : *#A↲
   1: *#T↲   (T) コマンドと"#"記号で，すべてのラインをタイプアウトする
   1:  fibonacci:
   2:      proc options(main);
   3:      dcl i fixed;
   4:      do i = 0 to 100;
   5:      put list(fib(i));
   6:      end;
   7:
   8:      fib:
   9:          proc(n) returns(fixed) recursive;
  10:          dcl n fixed;
  11:          if n = 0 then
  12:              return(1);
  13:          if n = 1 then
  14:              return(1);
  15:          return(fib(n-1) + fib(n-2));
  16:          end fib;
  17:      end fibonacci;
   1: *
```

Figure-11.4.9 バッファの内容をすべてタイプする

このようにエディット・バッファにロードされているテキストが全部タイプアウトされています．

さて，Fig-11.4.9の後に続けて，主要コマンドの実習を行います．コマンドを与える下線部と，それに対する ED の応答をよくご覧下さい．それらのサンプルランを Fig-11.4.10に示します．いくつかのコマンドをまとめて記述する様子や，「↲」，「0↲」などの使い方，「0TT↲」などの メタ・コマンド★の使い方に注目して下さい．

★ メタ・コマンド――他のコマンドに伴って使われるコマンドを言う．0TT は 0LT でもよい．

```
 1: *9:↵----- CPをライン9の頭にセット.
 9: *t↵ ----CPからラインの終りまでタイプ.
 9:          proc(n) returns(fixed) recursive;
 9: *5:t↵----CPをライン5の頭にセットし，そのラインをタイプ.（1行のコマンドで記述）
 5:       put list(fib(i));
 5: *3l↵---- CPを3ライン進め，その頭にセット.（注. 3lではなく3L）
 8: *t↵ ---- CPからラインの終りまでタイプ.
 8:       fib:
 8: *-4l↵----CPを4ライン前の頭にセット.
 4: *t↵ -----タイプ.
 4:       do i = 0 to 100;
 4: *7lt↵----CPを7ライン進め，そのラインをタイプ.
11:          if n = 0 then
11: *4↵----- CPを4ライン進め，そのラインをタイプ.
15:          return(fib(n-1) + fib(n-2));
15: *-2↵---- CPを2ライン前にセットし，そのラインをタイプ.
13:          if n = 1 then
13: *↵ ----- CPを次のラインにセットしタイプする.
14:             return(1);
14: *↵         同 上
15:          return(fib(n-1) + fib(n-2));
15: *↵         同 上
16:          end fib;
16: *↵         同 上
17:       end fibonacci;
17: *b↵----- CPをエディット・バッファの先頭にセット.
 1: *5:t↵----ライン5をタイプ. CPはライン5の先頭.
 5:       put list(fib(i));
 5: *slist^ZABCDEFG↵---- Sと^Zにより "list" を "ABCDEFG" に置き替え.
 5: *0↵----- そのラインをタイプ. CPはラインの頭にセットされる.（0LTの略コマンド）
 5:       put ABCDEFG(fib(i));
 5: *sfib^Z12345^Z0tt↵-----同様に "fib" を "12345" に置き替え，そのラインをタイプ.
 5:       put ABCDEFG(12345(i));
 5: *-↵------1の1を省略した形. 1ライン前にCPをセットしタイプ.（1は他のコマンドでも省略可）
 4:       do i = 0 to 100;
 4: *s0 to^Z↵------ Sと^Zの応用. "0_to" を削除する.
 4: *0↵-----そのラインをタイプ.
 4:       do i =  100;
 4: *10:t↵-----ライン10にCPをセットしタイプ.
10:          dcl n fixed;
10: *k↵-----CPのあるラインを削除.
10: *8::12t↵--------ライン8~ライン12をタイプ. CPはライン8の頭にセット.
 8:       fib:
 9:          proc(n) returns(fixed) recursive;  ┐
10:          if n = 0 then                      │
11:             return(1);                      ├（削除後ラインの確認）
12:          if n = 1 then                      ┘
 8: *10:↵-----CPをライン10にセット.
10: *3kt↵-----CPのラインを含めて，3ライン分を削除し，次のラインをタイプ.
10:             return(1);
10: *8:#t↵-----ライン8~最終ラインをタイプ.
 8:       fib:
 9:          proc(n) returns(fixed) recursive;  ┐
10:             return(1);                      │
11:          return(fib(n-1) + fib(n-2));       ├（削除後ラインの確認）
12:          end fib;                           │
13:       end fibonacci;                        ┘
 8: *b#t↵----CPをエディット・バッファの先頭にセットし，すべてのラインをタイプ.
```

```
     1:  fibonacci:
     2:      proc options(main);
     3:      dcl i fixed;
     4:      do i =  100;
     5:      put ABCDEFG(12345(i));
     6:      end;
     7:                                                   (現在までの結果)
     8:      fib:
     9:          proc(n) returns(fixed) recursive;
    10:              return(1);
    11:          return(fib(n-1) + fib(n-2));
    12:          end fib;
    13:      end fibonacci;
     1: *ffib^Zt↲----CP以後の文字列"fib"を捜し，その文字列の最後にセットし，CPからラインの終りまで
onacci:                                                                          タイプ.
     1: *3ffib^Zt↲--- CP以後3度目に現われる文字列"fib"を捜し，上と同様のことをする.
(n-2));
    11: *b↲---------CPをエディット・バッファの先頭にセット.
     1: *mffib^Z0tt↲---- エディット・バッファ内のCP以後のすべての文字列"fib"を捜して，そのラインを
     1:  fibonacci:                                              タイプ.（マクロ・コマンド）
     8:      fib:
    11:          return(fib(n-1) + fib(n-2));---(最初の"fib"でタイプされている)
    11:          return(fib(n-1) + fib(n-2));---(2番目の      〃      )
    12:          end fib;
    13:      end fibonacci;

BREAK "#" AT ^Z------(バッファの最後まで行ったのでブレークがかかった)
    13: *b↲-----CPをエディット・バッファの先頭にセット.
     1: *msfib^ZFIB^Z0tt↲-----CP以後のすべての文字列"fib"を"FIB"に置き替え，そのラインを
     1:  FIBonacci:                                              タイプ.（マクロ・コマンド）
     8:      FIB:
    11:          return(FIB(n-1) + fib(n-2));
    11:          return(FIB(n-1) + FIB(n-2));     (上のmfコマンドのリストと比較して下さい)
    12:          end FIB;
    13:      end FIBonacci;

BREAK "#" AT ^Z-----(もう見つからないのでブレークがかかった)
    13: *11:t↲---ライン11にCPをセットしタイプ.
    11:          return(FIB(n-1) + FIB(n-2));
    11: *i↲-----インサート・モードに入る.
    11:  INSERT THIS LINE NOW !↲
    12:  CP/M Learning System -I-↲     この2ラインをキーインした.
    13:  ^Z------^Zをキーインしてインサート・モードを終了する.
    13: *b#t↲----CPをエディット・バッファの先頭にセットし，全部のラインをタイプする.
     1:  FIBonacci:
     2:      proc options(main);
     3:      dcl i fixed;
     4:      do i =  100;
     5:      put ABCDEFG(12345(i));
     6:      end;
     7:
     8:      FIB:
     9:          proc(n) returns(fixed) recursive;        (最終的なテキスト)
    10:              return(1);
    11:  INSERT THIS LINE NOW !
    12:  CP/M Learning System -I-
    13:          return(FIB(n-1) + FIB(n-2));
    14:          end FIB;
    15:      end FIBonacci;
     1: *e↲------編集された内容をディスクにセーブし，EDを終了する.

A> -----(CP/Mにもどる)
```

Figure-11.4.10 テキストのエディッティング

このように最後はEコマンドで,エディットした結果をディスクにセーブし,ED を終了して CP/M に戻っています.

エディット作業が終った新しいファイル,FIB.PLI を TYPE コマンドで確認してみましょう.Fig-11.4.11に示します.

```
A>TYPE FIB.PLI↲
FIBonacci:
    proc options(main);
    dcl i fixed;
    do i =  100;
    put ABCDEFG(12345(i));
    end;

    FIB:
        proc(n) returns(fixed) recursive;
            return(1);
INSERT THIS LINE NOW !
CP/M Learning System -I-
        return(FIB(n-1) + FIB(n-2));
        end FIB;
    end FIBonacci;

A>
```

Figure-11.4.11 エディット後のファイル

このサンプルランでは,すべてのコマンドを小文字で与えています.小文字を含んだテキストをエディットする場合,インサートのIコマンドや,文字列置き替えのSコマンドなどいくつかのものは,大文字で与えた場合は,小文字でキーインした文字列が大文字に変換されたりして,小文字を扱うことができなくなります.よって,小文字を含むテキストをエディットする場合は,

ED 内のコマンドはすべて小文字で

入力する方が,トラブルがなくてベターです.

次はVコマンドによるエディット・バッファの残り容量を知る方法と,XとRコマンドによるファイルの組み替えと,ディスク上にあるライブラリ・ファイルの挿入について実習します.

その前に,ED コマンドの最初に作成した Fig-11.4.3のファイル名を,ライブラリ・ファイルを表わす".LIB"エクステンションにリネームしておきます.これはライブラリ・ファイル挿入の実習の準備です.そのサンプルランを Fig-11.4.12に示します.

```
A>REN FILEMAKE.LIB=FILEMAKE.TXT↲
A>
```

Figure-11.4.12 "LIB" エクステンションにリネーム

さて再び Fig-11.4.11に示すファイル，FIB.PLI に対し ED を起動します．サンプルランを Fig-11.4.13に示しますが，ここでも下線部のキーインと，その応答をよく対比してご覧下さい．

```
A>ED FIB.PLI↲
    : *0v↲ -----エディット・バッファの，空きエリアのバイト数/全エリアのバイト数を表示する．CP/M
27574/27575                               のサイズにより増減する．これは48K CP/Mの場合．
    : *#a↲ -----ファイルのすべてのラインをエディット・バッファにアペンドする．
   1: *0v↲ -----もう一度0Vコマンドを．空きエリアのバイト数が少し減ったことに注意．
27258/27575
   1: *7t↲ -----CPのラインから7ライン分をタイプする．
   1:  FIBonacci:
   2:       proc options(main);
   3:       dcl i fixed;
   4:       do i =  100;
   5:       put ABCDEFG(12345(i));
   6:       end;
   7:
   1: *7x↲ ----- CPから7ライン分を一時的にディスクにセーブする．
   1: *7kt↲ ----同じ7ライン分を削除し，次のラインをタイプする．
   1:       FIB:
   1: *#t↲ ----- CP後のすべてのラインをタイプする．
   1:       FIB:
   2:            proc(n) returns(fixed) recursive;
   3:                 return(1);
   4:  INSERT THIS LINE NOW !                          （7ライン分が削除されているこ
   5:  CP/M Learning System -I-                              とが確認される）
   6:            return(FIB(n-1) + FIB(n-2));
   7:            end FIB;
   8:       end FIBonacci;
   1: *-b↲ ------CPをアペンドしたラインの最終にセットする．
    : *r↲ ------先程セーブした内容を，CPの後に挿入する．
    : *b#t↲ ---- エディット・バッファのすべてのラインをタイプする．
```

```
  1:      FIB:
  2:          proc(n) returns(fixed) recursive;
  3:              return(1);
  4:  INSERT THIS LINE NOW !
  5:  CP/M Learning System -I-
  6:          return(FIB(n-1) + FIB(n-2));
  7:          end FIB;
  8:      end FIBonacci;
  9:  FIBonacci:
 10:      proc options(main);
 11:      dcl i fixed;
 12:      do i =  100;
 13:      put ABCDEFG(12345(i));
 14:      end;
 15:
  1: *9:t↲---------ライン9の頭にCPをセットし，そのラインをタイプする.
  9:  FIBonacci:
  9: *rFILEMAKE↲---- ディスク上のファイル “FILMAKE.LIB” をCPの後に挿入する.
 17: *b#t↲----- エディット・バッファのすべてのラインをタイプする.
  1:      FIB:
  2:          proc(n) returns(fixed) recursive;
  3:              return(1);
  4:  INSERT THIS LINE NOW !
  5:  CP/M Learning System -I-
  6:          return(FIB(n-1) + FIB(n-2));
  7:          end FIB;
  8:      end FIBonacci;
  9:        CAN'T HELP FALLING IN LOVE
 10:
 11:  Wise men say only fools rush in
 12:  But I can't help falling love with you
 13:  Shall I stay would it be a sin
 14:  If I can't help falling love with you
 15:
 16:             . . . . . . song by E.PRESLEY
 17:  FIBonacci:
 18:      proc options(main);
 19:      dcl i fixed;
 20:      do i =  100;
 21:      put ABCDEFG(12345(i));
 22:      end;
 23:
  1: *e↲ ------ディスクにセーブし，EDを終了する.

A>
```

（入れ替っていることが確認される）

（このように3つのブロックが編集された）

Figure-11.4.13 テキストのエディッティング

このように最後はEコマンドで終了します．この最後のEコマンドを忘れて，ディスケットを取り出したり，Ctrl-C でリブートしてしまったりすると，せっかく苦労してエディットしたテキストを，ディスクにセーブできなくなり，〝泣く〞ことになります．くれぐれも最後のEをお忘れなく！

このエディット作業で，再度更新されたテキスト・ファイル，FIB.PLI を TYPE コマンドでタイプアウトして，結果を見てみましょう．Fig-11.4.14に示します．

```
A>TYPE FIB.PLI ↲
    FIB:
        proc(n) returns(fixed) recursive;
            return(1);
INSERT THIS LINE NOW !
CP/M Learning System -I-
        return(FIB(n-1) + FIB(n-2));
        end FIB;
    end FIBonacci;
      CAN'T HELP FALLING IN LOVE

Wise men say only fools rush in
But I can't help falling love with you
Shall I stay would it be a sin
If I can't help falling love with you

           . . . . . . song by E.PRESLEY
FIBonacci:
    proc options(main);
    dcl i fixed;
    do i =  100;
    put ABCDEFG(12345(i));
    end;


A>
```

Figure-11.4.14 エディット後のファイルの確認

このようにディスク上にある "LIB" エクステンションの付いたライブラリ・ファイルは，Rコマンドにより，エディット・バッファに組み込むことができます．

実習C	ディスク上のテキスト・ファイルをエディットすると，バックアップ・ファイルが作られる

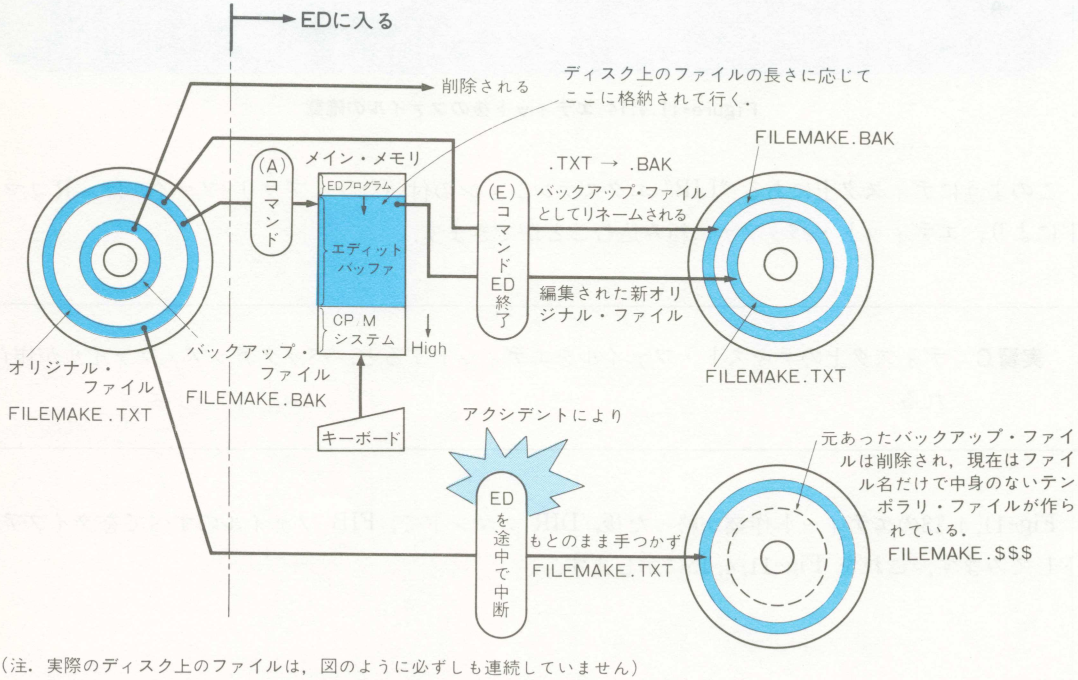
Fig-11.4.13のエディット作業が終った後，DIR コマンドで，FIB ファイルのすべてをタイプアウトしてみます．これを Fig-11.4.15に示します．

```
A>DIR FIB.*
A: FIB      PLI : FIB      BAK
A>
```

Figure-11.4.15 DIRコマンドによるファイルの確認

このように Fig-11.4.14で示した〝FIB.PLI〟ともう一つ,〝FIB.BAK〟が存在しています.このFIB.BAKを TYPE コマンドで内容を見てみると,この回で ED を起動する前のオリジナルのテキスト・ファイル(Fig-11.4.11に示されているもの)であることが分ります.

このように,ED を正常に終了すると,ED を行う前のオリジナル・ファイルは,エクステンションを〝.BAK〟とリネームされて,バックアップ用として,保存されるのです.このメカニズムを図示したものを Fig-11.4.16に示します.



(注.実際のディスク上のファイルは,図のように必ずしも連続していません)

Figure-11.4.16 既存のファイルをエディットする場合の流れ

実習D CP (Character Pointer) について

CP/M のエディタはこの CP 方式によるエディタです．スクリーン・エディタであれば目に見えるカーソルを基に各種のエディット処理を行いますが，CP/M のエディタは，目に見えないイメージ上の〝カーソル〟である CP を基に各種の処理を行います．

ED が起動して，テキストをエディット・バッファにアペンドした直後は，CP は最初のラインの頭にセットされています．

```
1：  *ABCDEFGHIJKLMN
      ↑
      CP
```

例えば，このような状態であるわけです．これをCコマンドの，

```
     *6C↲        を実行すると，
1：  *ABCDEFGHIJKLMN
            ↑
            CP
```

このように6文字分移動します．そしてその後のコマンドは，この CP を中心に処理が行われることになります．

よってこの状態でTコマンド (CP からラインの終りまでをタイプアウト) を実行すると

```
     *T↲
GHIJKLMN
```

とタイプアウトされます．

ED のコマンド一覧表

ED コマンドの簡単な一覧表を次の Fig-11.4.17 に示します．

EDコマンド	機　　　　能
nA	エディット・バッファにディスクからテキストをnライン分ロードする.
±B	CPをバッファの先頭/最後にセットする.
±nC	CPをn文字分前進/バックする.
±nD	CPから±n文字分を削除する.(注：上の±nCと同じく，字数を正確に数えるのはかなり面倒. 実際にはSコマンドなどで代用し，単純な場合を除き使用しない方がよい. 実習例参照)
E	バッファのテキストをディスクにセーブし，EDを終了する.
nF	CP以後で，一致する文字列のn番目のものを捜す.
H	一旦EDを終了し，再びEDに入る.
I	インサート・モードに入る.
nJ	条件が一致する文字列のn番目に，3種類の文字列を並べる.
±nK	CPのあるラインから±nライン分を削除する.
±nL	CPを現在のラインから±nライン分移動する.
nM	n回同じことをくり返す. マクロ・コマンド.
nN	n番目に表われる文字列を，アペンドされていないディスク上のテキストも対象に捜す.
O	一切を中断して，最初にEDを起動した状態にもどる.
±nP	CPから±nページ分(1ページは24ライン)をコンソールにプリントアウトする.
Q	EDを中止する. オリジナル・ファイルは変更されない.
R	Xコマンドによるテンポラリ・ファイルや，ディスク上のライブラリ・ファイルをバッファに挿入する.
nS	CP以後n番目に現われる文字列を任意の文字列と置き替える.
±nT	CPから±nライン分をタイプアウトする.
±U	+Uは小文字→大文字の変換を行わせる. −Uはその中止.
±V	+VはラインNo.の表示モードにする（version 2.2は最初から表示モード）. −VはラインNo.の表示をしないモードにする.
nW	バッファのテキストをnライン分ディスクにセーブする.
±nX	CPから±nライン分，テンポラリ・ファイルにセーブする. たいていはその後にRコマンドを使う.
±n	CPを±nライン移動し，そのラインをタイプアウトする.

注　意）☆±nの+の場合は+記号を省略できます.
☆nが1の時，この1は省略できます.
☆各コマンドはシリーズに並べて記述できます. 例 B6LTなど.
☆テキストに小文字を使用する場合，コマンドを小文字で入力する必要のあるものがあります（すべてべて小文字を使えばよい).

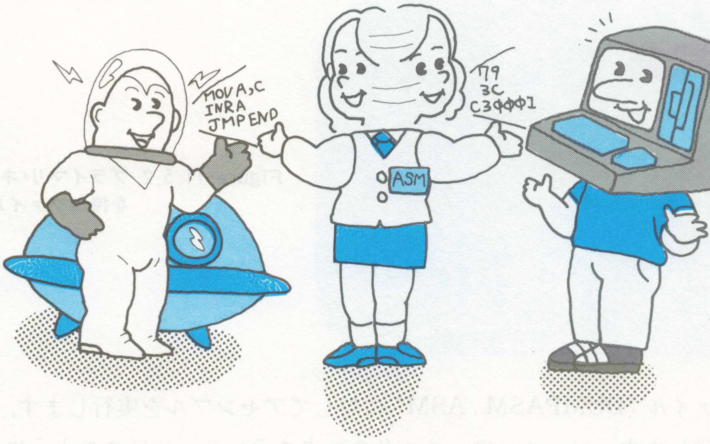
Figure-11.4.17 EDのコマンド一覧表

11.5 ASM（アセンブラ）（8080アセンブラ）

—— ASseMbler ——

機能　CP／M のアセンブル・プログラム〝ASM〟はインテル形式の8080のアセンブリ言語のソース・ファイルをアセンブルし，インテル HEX 形式のオブジェクト・ファイルと，メモリ・アドレスやオブジェクト・コード，エラーメッセージなどが含まれるリスト形式のプリント・ファイルを生成します．

（8080のマシン語は，Z80上でもそのまま実行できることは読者もよくご存知のことでしょう．）



実習A	CP／M に付属の DUMP プログラムのソース・ファイル〝DUMP.ASM〟をアセンブルする

まず DUMP.ASM を REN コマンドで〝DUMPASM.ASM〟とリネームし，後で区別できるようにしておきます．

このアセンブル作業は，すべてドライブA上で行うことにします．よって ASM プログラムの〝ASM.COM〟と，リネームした〝DUMPASM.ASM〟はドライブA上にあるとします．

リネームのサンプルランと，DIRで〝DUMPASM〟に関するファイルの確認を Fig-11.5.1～2に示します．

```
A>REN DUMPASM.ASM=DUMP.ASM↲
A>
```

Figure-11.5.1 最初にリネームしておく

```
A>DIR DUMPASM.*↲
A: DUMPASM  ASM
A>
```

Figure-11.5.2 プライマリ・ネーム〝DUMPASM〟を持つファイルを見る

さて，ソース・ファイル〝DUMPASM.ASM〟に対してアセンブルを実行します．そのサンプルランを Fig-11.5.3に示します．ソース・ファイル名のエクステンションはこのように必ず〝ASM〟でなければなりません．しかし，ASM を実行させるコマンド・ラインには，この〝ASM〟を書いてはいけません．このエクステンションの部分は後述の別の目的に使います．

```
A>ASM DUMPASM↲
CP/M ASSEMBLER - VER 2.0
0257
002H USE FACTOR
END OF ASSEMBLY

A>
```

Figure-11.5.3 アセンブラを起動する

このように，アセンブルに関するパラメータとメッセージが出力されて，アセンブルが終了しました．エラー・メッセージは出力されていませんので，エラーはなく，アセンブルできたことを示しています．出力メッセージの中の0257と言うのは，このオブジェクト・プログラムがメモリにロードされた場合のアドレスの最後(ユーザーが他のプログラムで使用してもよいエリアの始まり)を HEX で示しています．

さて，アセンブル終了により，どのようなファイルが生成されているでしょう．DIR コマンドで見てみます．これを Fig-11.5.4に示します．

```
A>DIR DUMPASM.*↲
A: DUMPASM  ASM : DUMPASM  PRN : DUMPASM  HEX
A>
```

Figure-11.5.4 アセンブル後のファイルを見る

このように，HEX と PRN 形式のファイルが出来ていることが確認されます．実際にどのようなファイル内容なのか，TYPE コマンドで見てみましょう．"DUMPASM.HEX" を Fig-11.5.5に，"DUMPASM.PRN" を Fig-11.5.6に示します．

```
A>TYPE DUMPASM.HEX↲

:10010000210000392215023157O2CDC101FEFFC2B4
:100110001B0111F301CD9C01C351013E80321302 3A
:10012000210000E5CDA201E1DA5101477DE60FC2D1
:100130004401CD7201CD59010FDA51017CCD8F01FF
:100140007DCD8F01233E20CD650178CD8F01C32366
:1001500001CD72012A1502F9C9E5D5C50E0BCD05F1
:1001600000C1D1E1C9E5D5C50E025FCD0500C1D101
:10017000E1C93E0DCD65013E0ACD6501C9E60FFE20
:100180000AD28901C630C38B01C637CD6501C9F5D6
:100190000F0F0F0FCD7D01F1CD7D01C90E09CD05EA
:1001A00000C93A1302FE80C2B301CDCE01B7CAB373
:1001B0000137C95F16003C3213022180000197EB757
:1001C000C9AF327C00115C000E0FCD0500C9E5D52A
:1001D000C5115C000E14CD0500C1D1E1C946494CE2
:1001E000452044554D502056455253494F4E2031DD
:1001F0002E34240D0A4E4F20494E505554204649 66
:100200004C452050524553454E54204F4E204449B2
:03021000534B2429
:0000000000

A>
```

ロード・アドレス部　　データ部　　チェックサム部

Figure-11.5.5 DUMPコマンドによりHEXファイルをダンプする

インテル HEX 形式のファイルは，このように16バイトづつのブロックになっており，それぞれのブロックの頭には，それがロードされるべきロード・アドレスや，ブロックの最後には16バイト毎のチェックサム・コードが付いています．

この HEX オブジェクト・ファイルは次の項で解説する LOAD プログラムで，実行可能な純マシン・コードに変換できます．

```
A>TYPE DUMPASM.PRN

                ;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
                ;
                ;       COPYRIGHT (C) 1975, 1976, 1977, 1978
                ;       DIGITAL RESEARCH
                ;       BOX 579, PACIFIC GROVE
                ;       CALIFORNIA, 93950
                ;
0100                    ORG     100H
0005 =          BDOS    EQU     0005H   ;DOS ENTRY POINT
0001 =          CONS    EQU     1       ;READ CONSOLE
0002 =          TYPEF   EQU     2       ;TYPE FUNCTION
0009 =          PRINTF  EQU     9       ;BUFFER PRINT ENTRY
000B =          BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
000F =          OPENF   EQU     15      ;FILE OPEN
0014 =          READF   EQU     20      ;READ FUNCTION
                ;
005C =          FCB     EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
0080 =          BUFF    EQU     80H     ;INPUT DISK BUFFER ADDRESS
         .
         .
         .
         .
013C 7C                 MOV     A,H
013D CD8F01             CALL    PHEX
0140 7D                 MOV     A,L
0141 CD8F01             CALL    PHEX
                NONUM:
0144 23                 INX     H       ;TO NEXT LINE NUMBER
0145 3E20               MVI     A,' '
0147 CD6501             CALL    PCHAR
014A 78                 MOV     A,B
014B CD8F01             CALL    PHEX
014E C32301             JMP     GLOOP
                ;
                FINIS:
                ;       END OF DUMP, RETURN TO CCP
                ;       (NOTE THAT A JMP TO 0000H REBOOTS)
0151 CD7201             CALL    CRLF
         .
         .
         .
         .
A>
```

Figure-11.5.6 プリント・ファイルを見る

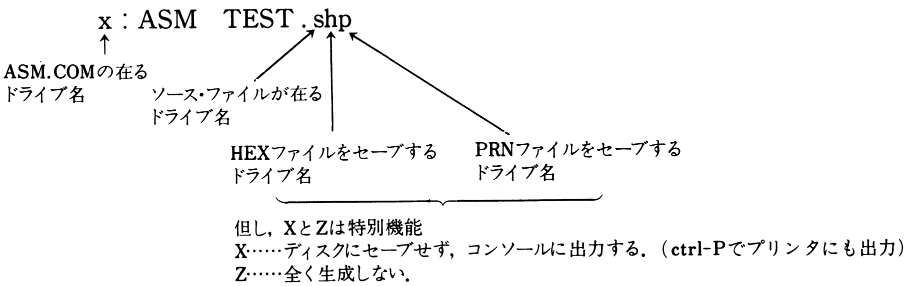
PRN ファイルは，ソース・ファイルの左端に，アドレスや，オブジェクト・コード，エラー・メッセージなどを付加したリスト・ファイルです．オリジナルの"DUMPASM．ASM"（＝"DUMP．ASM"）と比較してみて下さい．Fig-9.4.1のリスト参照．

CP／M のアセンブラは，マクロ機能や，リロケータブルなオブジェクト・コードを発生する機能はありませんがラベルは16文字まで識別し，複数のステートメントを１行に書くことや，"IF"，"END IF"文が使えるなど，いくつかの拡張機能があります．

ASM コマンドの書式について

Fig-11.5.3のサンプルランでは，ログイン・ディスク上のソース・ファイルをアセンブルし，HEXとPRN ファイルもディスクA上に生成しましたが，これらは次に示す書式で自由に選択できます．

ソース・ファイル "TEST．ASM" をアセンブルする場合のコマンド形式を次に示します．



但し，".shp"が省略された場合は，すべてにログイン・ディスクが指定されたとみなされます．
例えば，

A＞B：ASM TEST．ABX

このコマンドは，ドライブB上にある，ASM プログラムを起動して，ドライブA上のソース・ファイル "TEST．ASM" をアセンブルし，生成した HEX オブジェクト・ファイルをドライブB上にセーブし，PRN ファイルはコンソールに出力する．という意味になります．

11.6 LOAD（ロード）（HEXファイル ➡ COMファイル変換プログラム）

—— LOAD program ——

機能　インテル HEX 形式のオブジェクト・ファイル（CP／M のアセンブラによっても作られる）を，実行可能な純マシン・オブジェクト・ファイル（COM ファイル）に変換してディスク上にセーブする．

実習A　前項の ASM の実習で得られた〝DUMPASM．HEX〟を，COM ファイルに変換する

サンプルランを Fig-11.6.1に示します．

```
A>LOAD DUMPASM↲

FIRST ADDRESS 0100
LAST  ADDRESS 0212
BYTES READ    0113
RECORDS WRITTEN 03


A>
```

Figure-11.6.1 LOADプログラムの実行

このように〝DUMPASM〟とだけ書き，エクステンションの〝．HEX〟は書いてはいけません．もしこの HEX ファイルが他のドライブにある場合は，ファイル名の前に例によってドライブ名を指定する〝x：〟を付けます．

いくつかのパラメータが出力され，LOAD プログラムの実行が終了しました．COM ファイルが生成されていることを DIR コマンドで確認してみましょう．Fig-11.6.2に示します．

```
A>DIR DUMPASM.*↲
A: DUMPASM  PRN : DUMPASM  HEX : DUMPASM  COM : DUMPASM  ASM
A>
```

Figure-11.6.2 "DUMPASM" を持つファイルを見る

このように実行可能な COM ファイルが出来ています. では実際にこのプログラム(もともと DUMP プログラム)を実行してみましょう. 自分で自分自身の DUMPASM.COM をダンプしてみます. サンプルランを Fig-11.6.3に示します.

```
A>DUMPASM DUMPASM.COM↲

0000 21 00 00 39 22 15 02 31 57 02 CD C1 01 FE FF C2
0010 1B 01 11 F3 01 CD 9C 01 C3 51 01 3E 80 32 13 02
0020 21 00 00 E5 CD A2 01 E1 DA 51 01 47 7D E6 0F C2
0030 44 01 CD 72 01 CD 59 01 0F DA 51 01 7C CD 8F 01
0040 7D CD 8F 01 23 3E 20 CD 65 01 78 CD 8F 01 C3 23
0050 01 CD 72 01 2A 15 02 F9 C9 E5 D5 C5 0E 0B CD 05
0060 00 C1 D1 E1 C9 E5 D5 C5 0E 02 5F CD 05 00 C1 D1
0070 E1 C9 3E 0D CD 65 01 3E 0A CD 65 01 C9 E6 0F FE
0080 0A D2 89 01 C6 30 C3 8B 01 C6 37 CD 65 01 C9 F5
0090 0F 0F 0F 0F CD 7D 01 F1 CD 7D 01 C9 0E 09 CD 05
00A0 00 C9 3A 13 02 FE 80 C2 B3 01 CD CE 01 B7 CA B3
00B0 01 37 C9 5F 16 00 3C 32 13 02 21 80 00 19 7E B7
00C0 C9 AF 32 7C 00 11 5C 00 0E 0F CD 05 00 C9 E5 D5
00D0 C5 11 5C 00 0E 14 CD 05 00 C1 D1 E1 C9 46 49 4C
00E0 45 20 44 55 4D 50 20 56 45 52 53 49 4F 4E 20 31
00F0 2E 34 24 0D 0A 4E 4F 20 49 4E 50 55 54 20 46 49
0100 4C 45 20 50 52 45 53 45 4E 54 20 4F 4E 20 44 49
0110 53 4B 24 00 00 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

A>
```

Figure-11.6.3 "DUMPASM.COM" で自分自身をダンプする

このようにプログラムは完動しました. アセンブル・ソース・ファイルをアセンブルし, 生成された HEX ファイルを LOAD プログラムで COM ファイルに変換し, 実際に使えるプログラム・ファイルを作成する作業は大成功(?)だったわけです.

11.7 DDT (8080デバッガ)

ディーディーティー

—— Dynamic Debugging Tool ——

機能　8080マシン語のデバッガであり，次の機能があります．( ) 内はそのコマンド．

○COM ファイルを始め,すべてのタイプのファイルをメモリにロードする.(DDT または I と R)
○HEX ファイルを，そのロードアドレスに従ってメモリにロードする．(DDT または I と R)
○メモリのダンプ（D）
○メモリ内容の表示及び書き替え（S）
○メモリ・セグメントのブロック移動．(M)
○メモリの任意のエリアを任意のコードでうめる．（F）
○アセンブル機能．（A）
○逆アセンブル機能．（L）
○２つのブレーク・ポイントを設定してのプログラムの実行．（G）
○プログラムのトレース（シミュレーション)．（TまたはU）
○CPU 各レジスタ，フラグなどの表示・設定．（X）

などの強力な機能を持っています．

実習A　ファイルを，メモリにロードし，いくつかのコマンドを実行する

そのサンプルランを Fig-11.7.1に示します．DDT が起動すると，DDT のプロンプト記号〝−〞が表示されますので，その後に，コマンド・ラインをキーインします．

```
A>DDT DUMP.COM↲ ----- DDTを起動. 同時にDUMP .COMをアドレス100Hからのメモリにロードする.

DDT VERS 2.2
NEXT  PC
0300 0100
                                                                 それぞれのASCII表示.
                                                                 "." は標準ASCIIコード
                                                                 でないものを表わす.
-D100,17F↲----100H~17FHのメモリの内容をダンプする. (Dコマンド)
0100 21 00 00 39 22 15 02 31 57 02 CD C1 01 FE FF C2 !..9"..1W.......
0110 1B 01 11 F3 01 CD 9C 01 C3 51 01 3E 80 32 13 02 .........Q.>.2..
0120 21 00 00 E5 CD A2 01 E1 DA 51 01 47 7D E6 0F C2 !.........Q.G}...
0130 44 01 CD 72 01 CD 59 01 0F DA 51 01 7C CD 8F 01 D..r..Y...Q.|...
0140 7D CD 8F 01 23 3E 20 CD 65 01 78 CD 8F 01 C3 23 }...#> .e.x....#
0150 01 CD 72 01 2A 15 02 F9 C9 E5 D5 C5 0E 0B CD 05 ..r.*...........
0160 00 C1 D1 E1 C9 E5 D5 C5 0E 02 5F CD 05 00 C1 D1 .........._.....
0170 E1 C9 3E 0D CD 65 01 3E 0A CD 65 01 C9 E6 0F FE ..>..e.>..e.....

-L100,11A↲ -----100H~11AHを逆アセンブルする. (Lコマンド)
  0100  LXI  H,0000
  0103  DAD  SP
  0104  SHLD 0215
  0107  LXI  SP,0257
  010A  CALL 01C1
  010D  CPI  FF
  010F  JNZ  011B
  0112  LXI  D,01F3
  0115  CALL 019C
  0118  JMP  0151
  011B

-X↲ ----- CPUの現在の各状態を表示する. (Xコマンド)
C0Z0M0E0I0  A=00 B=0000 D=0000 H=0000  S=0100      P=0100      LXI  H,0000
 フラグ          各レジスタ              スタック・  プログラム・    現在の命令
                                        ポインタ    カウンタ
-T8↲ ---- 8ステップ分トレース実行する. (Tコマンド)
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI  H,0000
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0103 DAD  SP
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0100 S=0100 P=0104 SHLD 0215
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0100 S=0100 P=0107 LXI  SP,0257
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0100 S=0257 P=010A CALL 01C1
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0100 S=0255 P=01C1 XRA  A
C0Z1M0E1I0 A=00 B=0000 D=0000 H=0100 S=0255 P=01C2 STA  007C
C0Z1M0E1I0 A=00 B=0000 D=0000 H=0100 S=0255 P=01C5 LXI  D,005C*01C8

-S100↲------- 100Hからのメモリの内容の表示と置き替え. (Sコマンド)
0100 21 00↲
0101 00 01↲
0102 00 02↲
0103 39 03↲
0104 22 04↲   それぞれ左側に表示されている内容を,
0105 15 05↲   キーインした値に置き替える.
0106 02 06↲
0107 31 07↲
0108 57 08↲
0109 02 .↲----- ピリオドでSコマンドを終了する.

-D100,10F↲ ----- 100H~10FHのダンプ.
0100 00 01 02 03 04 05 06 07 08 02 CD C1 01 FE FF C2 ................
```

```
-A110↵------110Hからライン・アセンブルを始める.（Aコマンド）
0110  JMP 0020↵    ┐
0113  CALL 5000↵   │それぞれニーモニック, オペランド等をキーインする.
0116  HLT↵         ┘1ライン毎にアセンブルされ, オブジェクト・コードが出来て行く.
0117  ↵-----リターンのみキーインし, Aコマンドを終る.

-D110,11F↵----110H~11FHのダンプ. アセンブルの結果の確認.
0110 C3 20 00 CD 00 50 76 01 C3 51 01 3E 80 32 13 02 . ...Pv..Q.>.2..

-F120,14F,E5↵-----120H~14FHの間をE5Hのコードでうめる.（Fコマンド）

-D100,15F↵-------100H~15FHの間のダンプ. 結果の確認.
0100 00 01 02 03 04 05 06 07 08 02 CD C1 01 FE FF C2 ................
0110 C3 20 00 CD 00 50 76 01 C3 51 01 3E 80 32 13 02 . ...Pv..Q.>.2..
0120 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 ................
0130 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 ................
0140 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 ................
0150 01 CD 72 01 2A 15 02 F9 C9 E5 D5 C5 0E 0B CD 05 ..r.*...........

-IDUMP.ASM↵-----DUMP.ASMをファイル・コントロール・ブロックに登録する.（Iコマンド）
-R↵-------------登録されているファイルを100Hからメモリにロードする.（Rコマンド）
NEXT  PC        任意のアドレスにロードも可能.
1180 01C8

-D100,17F↵-------100H~17FHをダンプ. Rコマンドの確認.
0100 3B 09 46 49 4C 45 20 44 55 4D 50 20 50 52 4F 47 ;.FILE DUMP PROG
0110 52 41 4D 2C 20 52 45 41 44 53 20 41 4E 20 49 4E RAM, READS AN IN
0120 50 55 54 20 46 49 4C 45 20 41 4E 44 20 50 52 49 PUT FILE AND PRI
0130 4E 54 53 20 49 4E 20 48 45 58 0D 0A 3B 0D 0A 3B NTS IN HEX..;..;
0140 09 43 4F 50 59 52 49 47 48 54 20 28 43 29 20 31 .COPYRIGHT (C) 1
0150 39 37 35 2C 20 31 39 37 36 2C 20 31 39 37 37 2C 975, 1976, 1977,
0160 20 31 39 37 38 0D 0A 3B 09 44 49 47 49 54 41 4C  1978..;.DIGITAL
0170 20 52 45 53 45 41 52 43 48 0D 0A 3B 09 42 4F 58  RESEARCH..;.BOX

-G0↵-----0Hからの実行. 0Hを実行すると, リブートを起こす.（Gコマンド）
         任意のアドレスから, ブレーク・ポイントを設けて実行可能.
A>-------リブートしてCP/Mにもどっている.
```

Figure-11.7.1 DDTの実習

このように, まずファイル〝DUMP.COM〟をロードしたことに始り, いくつかの DDT コマンドを実行しましたが, 下線のあるキーインする部分と, その他の DDT が応答した部分をよく対比して, リストをご覧下さい.

また，DDT を起動するには，Fig-11.7.2に示すように，まず DDT のみを先に起動することもできます.

```
A>DDT↲
DDT VERS 2.2
-
```

Figure-11.7.2 DDTのみを起動

DDT が起動すると，このように DDT のプロンプト記号"－"が出力されますので，もし"DUMP.COM"をロードしたいのであれば，Fig-11.7.1のサンプルランにもある，I と R コマンドを使って，ロードすることが可能です.

11.8 DUMP（ディスク・ファイルのダンプ・プログラム）

ダンプ

—— DUMP program ——

DUMP プログラムは CP／M のコマンドと言うより，CP／M を応用してのプログラム作りの見本としての意味あいが強く，そのために，アセンブラ・ソース・ファイルが付いています.

機能　ディスク上の任意のファイルの内容を16進コードでダンプする．すべてのファイル形式に対して実行可能.

実習A　DUMP プログラムと他のリスト・アウト・コマンドとの比較

DUMP プログラムについては本書の随処に使われているので，説明の必要もないと思いますが，ここでは次の 3 つについて比較してみましょう.

1. DUMP プログラム
2. DDT でのダンプ（Dコマンド）
3. TYPE コマンド

さて，ダンプの対象となるファイルには，ドライブB上の BASIC のプログラム"TESTPRO.BAS"を使います．このファイルは，アスキー・セーブされているので，すべて ASCII キャラクタ・コードで構成されており，TYPE コマンドで表示することができます.

ではまず，DUMP プログラムでのダンプのサンプルランを Fig-11.8.1に示します.

```
A>DUMP B:TESTPRO.BAS↲

0000 31 30 20 27 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
0010 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
0020 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
0030 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 0D
0040 0A 32 30 20 27 0D 0A 33 30 20 27 20 20 20 20 20
0050 20 20 20 20 49 54 49 20 20 43 53 20 20 55 43 53
0060 20 20 46 4F 52 20 4D 4F 55 53 45 0D 0A 34 30 20
0070 27 0D 0A 35 30 20 27 3D 3D 3D 3D 3D 3D 3D 3D 3D
0080 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
0090 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
00A0 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D
00B0 3D 3D 0D 0A 36 30 20 27 0D 0A 37 30 20 50 52 49
00C0 4E 54 20 22 3D 3D 3D 3D 3D 20 44 49 53 43 52 49
00D0 4D 49 4E 41 54 45 44 20 41 56 4F 49 44 41 4E 43
00E0 45 2D 53 41 4D 45 20 50 52 4F 47 52 41 4D 20 57
00F0 49 54 48 20 34 20 42 4F 58 45 53 20 3D 3D 3D 3D
0100 3D 0D 0A 38 30 20 50 52 49 4E 54 0D 0A 39 30 20
      .
      .
      .
      .
```

Figure-11.8.1 DUMPプログラムによる表示

このように，アドレス表示は0000から始まり，アスキー表示部はありません．

次は DDT プログラムのD(ダンプ)コマンドによるダンプのサンプルランを Fig-11.8.2に示します.

```
A>DDT B:TESTPRO.BAS↲

DDT VERS 2.2
NEXT  PC
3400 0100

-D100,3400↲
0100 31 30 20 27 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 10 '============
0110 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D ================
0120 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D ================
0130 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 0D ===============.
0140 0A 32 30 20 27 0D 0A 33 30 20 27 20 20 20 20 20 .20 '..30 '
0150 20 20 20 20 49 54 49 20 20 43 53 20 20 55 43 53     ITI  CS  UCS
0160 20 20 46 4F 52 20 4D 4F 55 53 45 0D 0A 34 30 20   FOR MOUSE..40
0170 27 0D 0A 35 30 20 27 3D 3D 3D 3D 3D 3D 3D 3D 3D '..50 '=========
0180 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D ================
0190 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D ================
01A0 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D ================
01B0 3D 3D 0D 0A 36 30 20 27 0D 0A 37 30 20 50 52 49 ==..60 '..70 PRI
01C0 4E 54 20 22 3D 3D 3D 3D 3D 20 44 49 53 43 52 49 NT "===== DISCRI
01D0 4D 49 4E 41 54 45 44 20 41 56 4F 49 44 41 4E 43 MINATED AVOIDANC
01E0 45 2D 53 41 4D 45 20 50 52 4F 47 52 41 4D 20 57 E-SAME PROGRAM W
01F0 49 54 48 20 34 20 42 4F 58 45 53 20 3D 3D 3D 3D ITH 4 BOXES ====
0200 3D 0D 0A 38 30 20 50 52 49 4E 54 0D 0A 39 30 20 =..80 PRINT..90
          .
          .
          .
          .
```

Figure-11.8.2 DDTのDコマンドによる表示

このように DDT を起動すると,目的のファイルはアドレス 100H からにロードされるので,Dコマンドで100Hからダンプを開始するようキーインします.DDT のDコマンドでは,表示の右側に,同じ行の16バイトそれぞれに対応する ASCII 表示部があります.

最後はダンプではなく TYPE です．参考のため，アスキー・ファイルであるこの〝TESTPRO.BAS〟の内容をタイプアウトしておきます．それを Fig-11.8.3に示します．

```
A>TYPE B:TESTPRO.BAS ↵

10 '==========================================================
20 '
30 '          ITI  CS  UCS  FOR MOUSE
40 '
50 '==========================================================
60 '
70 PRINT "===== DISCRIMINATED AVOIDANCE-SAME PROGRAM WITH 4 BOXES =====
80 PRINT
90 '
100 '***** INITIAL SET *****
110 '
120 DEFINT A-Z     'DEFAIN ALL VALIABLES TO INTEGER
130 '
140 'DISK FILE NAME DEFINE
150 INPUT "FILE NAME";FILENAMEIN$
160 INPUT "DATE";DATEIN$
170 '
180 'DISK FILE OPEN
190 OPEN "R",#1,FILENAMEIN$
    .
    .
    .
    .
```

Figure-11.8.3 TYPEコマンドによる表示

以上の3つのリストをよく比較して，それぞれの特徴をつかんで下さい．

11.9 SUBMIT（サブミット）（バッチ処理プログラム）

—— SUBMIT program ——

機能　いくつかの CP／M コマンドやプログラムを，実行順にシーケンシャルに並べたファイルに従って，次々と自動的に実行させるバッチ処理を行う．



実習A　合計８つのビルトイン・コマンドや，トランジェント・プログラムからなる SUBMIT ファイルを作り，それらを順次自動的に実行する

まず，コマンドやプログラムを実行順に並べた〝SUB〞ファイルを，ED（エディタ）で作ります．その SUB ファイルを TYPE コマンドで Fig-11.9.1にリストしておきます．

```
A>TYPE BATCH.SUB↲

DIR DUMPSUB.*-----------------①
PIP DUMPSUB.ASM=DUMP.ASM-----②
DIR DUMPSUB.*-----------------③
ASM DUMPSUB-------------------④
DIR DUMPSUB.*-----------------⑤
LOAD DUMPSUB------------------⑥
STAT DUMPSUB.*----------------⑦
DUMPSUB DUMPSUB.COM-----------⑧

A>
```

Figure-11.9.1 〝BATCH.SUB〞の内容

SUB ファイルは，〝BATCH.SUB〞というファイル名にでもしておきます．これらのコマンドの一つ一つは簡単なものなので，すぐに理解できると思いますが，一応説明しますと，

1) ディスクA上のすべての〝DUMPSUB〞ファイルをタイプアウトする．
2) DUMP プログラムのアセンブル・ソース・ファイル〝DUMP.ASM〞のコピーを，同じディスク上に〝DUMPSUB.ASM〞としてセーブする．
3) すべての〝DUMPSUB〞ファイルをタイプアウトする．
4) 〝DUMPSUB.ASM〞をアセンブルする．
5) すべての〝DUMPSUB〞ファイルをタイプアウトする．
6) 4)のアセンブルによって生成された〝DUMPSUB.HEX〞を，純マシン・コードの〝COM〞ファイルに変換する．
7) すべての〝DUMPSUB〞ファイルに関して，STAT プログラムで調べる．
8) 〝DUMPSUB〞はダンプ・プログラムなので，実際に，自分で自分自身をダンプしてみる．

以上8つのコマンドラインを ED を使ってファイルして下さい．ファイル名は〝BATCH.SUB〞としておきましょう．この時エクステンションは必ず〝SUB〞でなければなりません．

SUBMITの実行

いよいよ SUBMIT を実行しますが，〝SUB〞ファイルは必ずドライブAになければなりません．〝SUB〞ファイル中で使用する各ファイルと〝SUBMIT.COM〞は，今回の例では，すべてドライブA上に用意しておかなければなりません．

実行は簡単です．Fig-11.9.2に示すように，SUBMIT コマンドを与えるだけで，あとはこのように，〝SUB〞ファイルに従って自動的に最後まで実行してくれます．

```
A>SUBMIT BATCH↲

A>DIR DUMPSUB.*--------①が実行される.
NO FILE
A>PIP DUMPSUB.ASM=DUMP.ASM-----②が実行される.

A>DIR DUMPSUB.*------③が実行される.
A: DUMPSUB  ASM
A>ASM DUMPSUB--------④が実行される.
CP/M ASSEMBLER - VER 2.0
0257
002H USE FACTOR
END OF ASSEMBLY

A>DIR DUMPSUB.*------⑤が実行される.
A: DUMPSUB  ASM : DUMPSUB  PRN : DUMPSUB  HEX
A>LOAD DUMPSUB-------⑥が実行される.

FIRST ADDRESS 0100
LAST  ADDRESS 0212
BYTES READ    0113
RECORDS WRITTEN 03

A>STAT DUMPSUB.*-----⑦が実行される.

 Recs  Bytes  Ext Acc
   33     5k    1 R/W A:DUMPSUB.ASM
    3     1k    1 R/W A:DUMPSUB.COM
    7     1k    1 R/W A:DUMPSUB.HEX
   60     8k    1 R/W A:DUMPSUB.PRN
Bytes Remaining On A: 74k

A>DUMPSUB DUMPSUB.COM-------⑧が実行される.

0000 21 00 00 39 22 15 02 31 57 02 CD C1 01 FE FF C2
0010 1B 01 11 F3 01 CD 9C 01 C3 51 01 3E 80 32 13 02
0020 21 00 00 E5 CD A2 01 E1 DA 51 01 47 7D E6 0F C2
0030 44 01 CD 72 01 CD 59 01 0F DA 51 01 7C CD 8F 01
0040 7D CD 8F 01 23 3E 20 CD 65 01 78 CD 8F 01 C3 23
0050 01 CD 72 01 2A 15 02 F9 C9 E5 D5 C5 0E 0B CD 05
0060 00 C1 D1 E1 C9 E5 D5 C5 0E 02 5F CD 05 00 C1 D1
0070 E1 C9 3E 0D CD 65 01 3E 0A CD 65 01 C9 E6 0F FE
0080 0A D2 89 01 C6 30 C3 8B 01 C6 37 CD 65 01 C9 F5
0090 0F 0F 0F 0F CD 7D 01 F1 CD 7D 01 C9 0E 09 CD 05
00A0 00 C9 3A 13 02 FE 80 C2 B3 01 CD CE 01 B7 CA B3
00B0 01 37 C9 5F 16 00 3C 32 13 02 21 80 00 19 7E B7
00C0 C9 AF 32 7C 00 11 5C 00 0E 0F CD 05 00 C9 E5 D5
00D0 C5 11 5C 00 0E 14 CD 05 00 C1 D1 E1 C9 46 49 4C
00E0 45 20 44 55 4D 50 20 56 45 52 53 49 4F 4E 20 31
00F0 2E 34 24 0D 0A 4E 4F 20 49 4E 50 55 54 20 46 49
0100 4C 45 20 50 52 45 53 45 4E 54 20 4F 4E 20 44 49
0110 53 4B 24 00 00 00 00 00 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

A>------- すべての実行が終了し，CP/Mにもどる．
```

Figure-11.9.2 SUBMITを実行する

このように，すべての実行が終って CP／M に戻っています．

最初のコマンド・ラインは，このように SUB ファイル名の〝.SUB〟エクステンションは書いてはいけません．

SUBMIT プログラムは，実習例のように，いくつかのコマンドや，プログラムを一括して自動的に実行できます．さらに CP／M の2.2では，〝XSUB〟というサブ・プログラムが加わって，この XSUB を〝SUB〟ファイルの最初に書いておくと，ED や，DDT 内のサブ・コマンドにも SUBMIT を適用することができます．

ここでの実習では，〝SUB〟ファイル（例として，BATCH.SUB を作った）に実際のファイル名を書きましたが，本来は実際のファイル名の代りに〝$1，$2，$3，……〟という記号を代用しておき，実行時のコマンド・ラインで，実際のファイル名を任意に設定できるのです．

この SUBMIT プログラムは，何度も同じ処理を繰り返さなければならないような使い方に応用すると非常に便利です．

11.10 SYSGEN（CP/Mシステム生成プログラム）
シスジェン

—— SYStem GENerator ——

機能　ディスク上の CP／M システム部を,他のディスクにコピーする.またディスケット上の CP／M システム部を,TPA にロードしたり,新しく作成したものや,変更を加えたりした TPA 上の CP／M イメージを,ディスクのシステム・トラックにセーブすることができる. CP／M の BIOS の変更や,メモリ・サイズの変更などに必ず使用される.

CP／M の〝システム部〟は, 6章でも述べたように,例えば8インチ標準ディスケットの場合,トラック0と1の2本のトラックに記録されています.この2本の〝システム用トラック〟には CP／M の本体が記録されていて,起動時にメイン・メモリ上に読み出され, CP／M の働きが開始されるのです.

この2本のシステム・トラックは,〝ファイル〟ではないので, PIP プログラムの対象にはなりません.よって,このトラックに対してリード／ライトを行うことができるのは,標準 CP／M には付属していない特別なコピー・プログラムなどを使わない限り, SYSGEN プログラムのみであるわけです.

本書での実習は,ディスケットからディスケットへの CP／M システムのコピーだけを行います.

実習A　ドライブAに挿入されているシステム・ディスケットの CP／M システム部を,ドライブB上のディスクにコピーする

この実習は, 8.7 章の「バックアップ・コピー」で行ったことと同じです.そのサンプルランを Fig-11.10.1に示します.

```
A>SYSGEN↲

SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)A
SOURCE ON A, THEN TYPE RETURN↲
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B
DESTINATION ON B, THEN TYPE RETURN↲
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)↲

A>
```

Figure-11.10.1 CP/Mシステムのコピー

これでドライブB上のディスケットに，CP/M システムが書き込まれ，このディスケットは今後，〝システム・ディスケット〟となったわけです．

サンプルランでのメッセージの SOURCE DRIVE NAME とは，オリジナル・ディスケットが挿入されるドライブ名のことであり，

DESTINATION DRIVE NAME とは,これから書き込みを行おうとするドライブ名のことです．この指定を誤ると，致命的なミスになりかねないので，十分に確認の上実行して下さい．

このドライブの指定の方法により，例えば，

ドライブA⟶ドライブB

ドライブA⟵ドライブB

ドライブA⟶ドライブA　（途中でディスケットを交換する）

ドライブB⟶ドライブB　（　　　〃　　　）

など，自由にコピーを行うことができます．

SYSGEN プログラムによるディスケットへの書込みに関しては,通常問題になる,ディスケットが交換されたこと（ドライブのフタを開けたこと）による自動的な書込禁止は無視されますので，ディスケットを交換しても，リブート操作をする必要はありません．

11.11 MOVCPM (CP/Mシステム・リロケート・プログラム)

ムーブシーピーエム

—— MOVe CP/M system ——

機能 CP/M システムのリロケート・プログラム．使用するコンピュータのメモリ・サイズに合わせて，CP/M サイズを変更する．

MOVCPM プログラムは，BIOS を除く CP/M システムと，それのリロケート・プログラムが一体となったものです．11.1章でも少し触れていますが，CP/M は起動すると，メイン・メモリの高位のアドレスに常駐します．しかし，このアドレスは使用目的や，マシンによって異り，一定ではありません．

例えば，アドレスC000H以上に，別のモニタ ROM や，特別のプログラム・エリアを持っていた場合，その CP/M サイズの最大は48Kということになります．これらの様々の状況に適合させるため，ユーザーが自由に簡単な操作で CP/M システムをリロケートできるように作られたのが，この MOVCPM プログラムであるわけです．

MOVCPM についての実習は本書では行わず，続巻〝実習 CP/M〟で，解説・実習を行います．

ログイン・ディスクとオブジェクト・ディスク

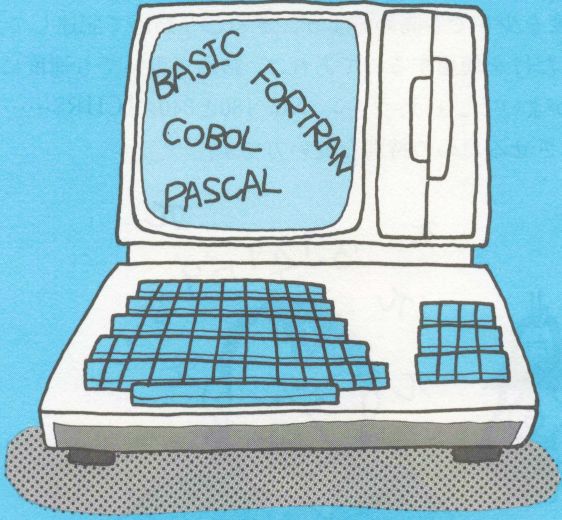
ログイン・ディスクと，実行しようとするプログラムが存在するディスク，それに，そのプログラムによりアクセスされるディスクとの，3者の関係を理解することは，CP/M を使いこなす上で大切なことです．ここで，すでに学んだ STAT プログラムを例に，これらの関係をサンプルランとして示しておきましょう．Fig-11.3.8も同じ様な使い方ですから，参考にして下さい．

```
A>B:↲ ------------ 実習のためログイン・ディスクをBに.
B>STAT DUMP.ASM↲---- STATを実行.
STAT? ----------- ディスクBにはSTATプログラムがない.

B>A:STAT DUMP.ASM↲--- ディスクA上のSTATを実行.
File Not Found ---- STATプログラムは実行されたが，B上にDUMP.ASMがない.

B>A:STAT A:DUMP.ASM↲---DUMP.ASMにもA:を付けて実行.
 Recs  Bytes  Ext Acc
   33     5k    1 R/W A:DUMP.ASM     } ログイン・ディスクBから，A上のSTATを実行し，
Bytes Remaining On A: 106k           } A上のDUMP.ASMがレポートされた.
B> ---(ログイン・ディスクがAであれば，2行目のコマンドで実行できた)
```

12章　高級言語を使ってCP/Mを理解する



BASICコンパイラによる実行
MBASICインタプリタによる実行
MBASICとコンパイラの比較

これまで，標準 CP／M に付属しているコマンド，及びプログラムについて実習してきましたが，今度は，CP／M をとりまく豊富なソフトウェアのうち最も一般的な BASIC 言語を CP／M 上で使って実習してみます．

円周率πの1000桁をマイクロソフト社の BASIC-80 (MBASIC) と，BASIC コンパイラを使って求めてみました．コンパイラを使っても思ったほど高速にはなりませんでしたが，それでもインタプリタより 4 倍以上は速くなっています．

12.1　BASICコンパイラによる実行

πの1000桁を求める BASIC 言語でのプログラム★を Fig-12.1.1に TYPE コマンドで示します．インタプリタでの実行速度を少しでも高めるよう，ぎっしり詰めて記述してあり，読みづらいと思います．BASIC コンパイラだけを使用するのであれば，詰めて書いても速度には関係しないので，もっと分りやすく記述した方がよいでしょう．ライン No. 180と340の「CHR$……」というのは，if800 CP／M での日付・時刻を表示させるための特別な使い方です．



★ プログラム——このプログラムはアスキー出版発行の PC-8001 BASIC ゲームブックの林氏のπの計算を一部変更して作ったものです．

```
A>TYPE PI1000.BAS

100 '     ---------------------------------------------------------
110 '     =====  CP/M Learning System -I-  DEMO Program  =====
120 '     =====                                          =====
130 '     =====             PI 1000 Figures              =====
140 '     ---------------------------------------------------------
150 '
160 DEFINT A-Z: DEFSNG R,M,X
170 DIM A(252),B(252),C(252),D(4),F(4)
180 PRINT: PRINT "START TIME = "; CHR$(&H1B); "A"
190 GOTO 260
200 GOSUB 210: FOR I=1 TO 252: A(I)=B(I): NEXT: FOR J=1 TO 252: B(J)=0: NEXT: RE
TURN
210 R=0: FOR I=1 TO 252: X=A(I)+R*10000: B(I)=INT(X/M): R=X-B(I)*M: NEXT: RETURN
220 D=0: FOR I=252 TO 1 STEP -1: C(I)=C(I)-B(I)-D: D=0: IF C(I)<0 THEN C(I)=C(I)
+10000: D=1
230 NEXT: RETURN
240 C=0: FOR I=252 TO 1 STEP -1: C(I)=C(I)+B(I)+C: C=0: IF C(I)>=10000 THEN C(I)
=C(I)-10000: C=1
250 NEXT: RETURN
260 GOSUB 420
270 A(1)=80: FOR O=1 TO 1494 STEP 2
280 M=25: GOSUB 200: M=O: GOSUB 210: GOSUB 240: O=O+2
290 M=25: GOSUB 200: M=O: GOSUB 210: GOSUB 220: NEXT
300 FOR J=1 TO 252: A(J)=0: B(J)=0: NEXT: A(1)=956
310 FOR O=1 TO 421 STEP 2
320 M=239: GOSUB 200: M=239: GOSUB 200: M=O: GOSUB 210: GOSUB 220: O=O+2
330 M=239: GOSUB 200: M=239: GOSUB 200: M=O: GOSUB 210: GOSUB 240: NEXT
340 PRINT: PRINT "END TIME   = "; CHR$(&H1B); "A": PRINT
350 PRINT "PI= 3.";
360 I=2
370 PRINT " ";: Y=C(I): GOSUB 430: I=I+1: IF I=252 THEN 400
380 IF (I-2) MOD 10=0 THEN PRINT: PRINT TAB( 7);
390 GOTO 370
400 PRINT: PRINT
410 END   '*** END OF PROGRAM ***.
420 F(1)=1000: F(2)=100: F(3)=10: F(4)=1: RETURN
430 FOR L=1 TO 4: D(L)=INT(Y/F(L)): Y=Y MOD F(L): NEXT
440 FOR L=1 TO 4: Z=D(L): ON Z+1 GOTO 450,460,470,480,490,500,510,520,530,540
450 PRINT "0";: GOTO 550
460 PRINT "1";: GOTO 550
470 PRINT "2";: GOTO 550
480 PRINT "3";: GOTO 550
490 PRINT "4";: GOTO 550
500 PRINT "5";: GOTO 550
510 PRINT "6";: GOTO 550
520 PRINT "7";: GOTO 550
530 PRINT "8";: GOTO 550
540 PRINT "9";
550 NEXT: RETURN
560 '
570 'END OF LIST

A>
```

Figure-12.1.1 BASICソース・プログラム

さて，この BASIC のソース・プログラムを，CP／M のエディタ（ED）か，あるいは MBASIC があるなら，MBASIC を起動して，その BASIC 上で作成します．ED で作成する場合，ファイル名のエクステンションには必ず〝BAS〟を付けて下さい．MBASIC 上で作成する場合は，エクステンションの〝BAS〟は自動的に付きますが，ディスクにセーブする時はアスキー・セーブで行って下さい．アスキー・セーブでないと，BASIC コンパイラは，ソース・ファイルとして受け付けません．

Fig-12.1.1に示したソース・プログラムのファイルが出来上ったとしましょう（ファイル名は〝PI1000.BAS〟としました）．MBASIC と BASIC コンパイラは，ソース・レベルでコンパチブルですから，MBASIC ではこのままでも実行することができますが，先にコンパイルの作業をやってしまいましょう．コンパイルには〝BASIC COMPILER〟のディスケットに含まれているファイルの中から，Fig-12.1.2のDIR で示す各ファイルが必要です．

```
A>DIR↲
A: BASCOM    COM : BASLIB   REL : L80       COM : PI1000   BAS
A>
```

Figure-12.1.2 コンパイルに必要なファイル

では現在のログイン・ディスク上に，これらのファイルが在るものとして，作業を始めます．

まず，ソース・ファイルの〝PI1000.BAS〟を〝BASCOM.COM〟によりコンパイルします．そのサンプルランを Fig-12.1.3に示します．

```
A>BASCOM PI1000,PI1000=PI1000↵

00000 Fatal Error(s)
14125 Bytes Free

A>
```

Figure-12.1.3 コンパイル

コンパイル時のコマンドの書き方により，種々の機能がありますが，この例の書き方が最も標準的なもので，コマンドの終了により，コンパイルされた〝PI1000.BAS〟のリロケータブルなオブジェクト・ファイル，〝PI1000.REL〟と，PRN 形式のリスト・ファイルが生成されます．

このようにコンパイルのメッセージが出力されて，コンパイルが終了しました．コンパイル・エラーはなく，CP／M 上の未使用エリアのバイト数が10進で表示されています．この値は，CP／M システムのメモリ・サイズによって異ります．もちろんコンパイルするプログラムによっても異ります．

生成されたファイルを DIR で確認してみましょう．Fig-12.1.4に示します．

```
A>DIR PI1000.*↵
A: PI1000   BAS : PI1000   PRN : PI1000   REL
A>
```

Figure-12.1.4 コンパイル後のファイルの確認

このように，REL と PRN 形式のファイルが生成されています．参考までに〝PI1000.PRN〟を TYPE コマンドで見てみましょう．Fig-12.1.5に示しますが，コンパイルの様子が分ります．例えばライン No.190の GOTO 260は，JMP L00260 とマシン語に置き換えられています．

```
A>TYPE PI1000.PRN↵

BASCOM 5.23 - Copyright 1979, 80 (C) by MICROSOFT - 11732 Bytes Free
 0014 0007      100 '      ---------------------------------------------------
        ** 0014'        CALL    $5.0
        ** 0017'L00100:
 0017 0007      110 '      =====  CP/M Learning System -I-  DEMO Program  =====
        ** 0017'L00110:
 0017 0007      120 '      =====                                          =====
        ** 0017'L00120:
 0017 0007      130 '      =====             PI 1000 Figures              =====
        ** 0017'L00130:
 0017 0007      140 '      ---------------------------------------------------
        ** 0017'L00140:
 0017 0007      150 '
        ** 0017'L00150:
 0017 0007      160 DEFINT A-Z: DEFSNG R,M,X
        ** 0017'L00160:
 0017 0007      170 DIM A(252),B(252),C(252),D(4),F(4)
        ** 0017'L00170:
 0017 0609      180 PRINT: PRINT "START TIME = "; CHR$(&H1B); "A"
        ** 0017'L00180: CALL    $PR0A
        ** 001A'        LXI     H,<const>
        ** 001D'        CALL    $PV2D
        ** 0020'        CALL    $PR0A
        ** 0023'        LXI     H,<const>
        ** 0026'        CALL    $PV1D
        ** 0029'        LXI     H,001B
        ** 002C'        CALL    $CHR
        ** 002F'        CALL    $PV1D
        ** 0032'        LXI     H,<const>
        ** 0035'        CALL    $PV2D
 0038 0609      190 GOTO 260
        ** 0038'L00190: JMP     L00260
 003B 0609      200 GOSUB 210: FOR I=1 TO 252: A(I)=B(I): NEXT: FOR J=1 TO 252: B(J)=0: NEXT: RETURN
        ** 003B'L00200: CALL    L00210
        ** 003E'        LXI     H,0001
        ** 0041'        JMP     I00000
        ** 0044'I00001:
        ** 0044'        LHLD    I%
        ** 0047'        DAD     H
        ** 0048'        PUSH    H
     .
     .
```

```
  .
  .
  .
0503 0625      530 PRINT "8";: GOTO 550
      ** 0503'L00530: CALL    $PROA
      ** 0506'        LXI     H,<const>
      ** 0509'        CALL    $PV1D
      ** 050C'        JMP     L00550
050F 0625      540 PRINT "9";
      ** 050F'L00540: CALL    $PROA
      ** 0512'        LXI     H,<const>
      ** 0515'        CALL    $PV1D
0518 0625      550 NEXT: RETURN
      ** 0518'L00550: LHLD    L%
      ** 051B'        INX     H
      ** 051C'I00032:
      ** 051C'        SHLD    L%
      ** 051F'        LHLD    L%
      ** 0522'        LXI     D,FFFB
      ** 0525'        MOV     A,H
      ** 0526'        RAL
      ** 0527'        JC      I00034
      ** 052A'        DAD     D
      ** 052B'        DAD     H
      ** 052C'I00034: JC      I00033
      ** 052F'        RET
0530 0625      560 '
      ** 0530'L00560:
0530 0625      570 'END OF LIST
      ** 0530'L00570:
0530 0625
      ** 0530'        CALL    $END
059F 063F

00000 Fatal Error(s)
14125 Bytes Free


A>
```

Figure-12.1.5 プリントファイルを見る

次は，生成されたオブジェクト〝PI1000.REL〟を，リンカーの LINK-80（〝L80.COM〟）で，ライブラリの〝BASLIB.REL〟とリンクして，実行可能なオブジェクトを生成する作業です．そのサンプルランを Fig-12.1.6に示します．

```
A>L80 PI1000/E,PI1000/N↵

Link-80  3.4  01-Dec-80  Copyright 1979,80 (C) Microsoft

Data    0103    3065    <12130>

22170 Bytes Free
[0742   3065        48]

A>
```

Figure-12.1.6 リンキング

このように各種のメッセージが出力されて，リンクが終了しました．［ ］内の数字は左から出来上った〝COM〟マシン語ファイルのプログラム部の開始アドレス（HEX），最終アドレス（HEX），全体の256バイトのページ数，を示しています．

リンカーで生成された〝COM〟オブジェクト・ファイルを含めて，すべての〝PI1000〟に関するファイルの様子を STAT コマンドで調べてみましょう．STAT プログラムはドライブB上にあるとします．それを Fig-12.1.7に示します．

```
A>B:STAT PI1000.*↵

 Recs  Bytes  Ext Acc
   16     2k    1 R/W A:PI1000.BAS
   95    12k    1 R/W A:PI1000.COM
  123    16k    1 R/W A:PI1000.PRN
   18     3k    1 R/W A:PI1000.REL
Bytes Remaining On A: 96k

A>
```

Figure-12.1.7 STATプログラムによりファイルの状態を見る

このように BASIC で書かれたソース・プログラム，〝BAS〟から，目的のマシン語のプログラム，〝COM〟が生成されました．あとはこのマシン語の〝PI1000.COM〟ファイルのみで，すべての CP／

M マシン上でπの1000桁が求められるわけです.

さて, 実行してみましょう. 時間を計測するのに便利なので if800 CP／M 上で実行します. 実行はプライマリ・ネームの〝PI1000〞だけをキーインしてリターンします. サンプルランを Fig-12.1.8に示します.

```
A>PI1000↲

START TIME =  81/08/03 MON 19:00:04

END TIME   =  81/08/03 MON 19:52:31

PI= 3. 1415 9265 3589 7932 3846 2643 3832 7950 2884 1971
       6939 9375 1058 2097 4944 5923 0781 6406 2862 0899
       8628 0348 2534 2117 0679 8214 8086 5132 8230 6647
       0938 4460 9550 5822 3172 5359 4081 2848 1117 4502
       8410 2701 9385 2110 5559 6446 2294 8954 9303 8196
       4428 8109 7566 5933 4461 2847 5648 2337 8678 3165
       2712 0190 9145 6485 6692 3460 3486 1045 4326 6482
       1339 3607 2602 4914 1273 7245 8700 6606 3155 8817
       4881 5209 2096 2829 2540 9171 5364 3678 9259 0360
       0113 3053 0548 8204 6652 1384 1469 5194 1511 6094
       3305 7270 3657 5959 1953 0921 8611 7381 9326 1179
       3105 1185 4807 4462 3799 6274 9567 3518 8575 2724
       8912 2793 8183 0119 4912 9833 6733 6244 0656 6430
       8602 1394 9463 9522 4737 1907 0217 9860 9437 0277
       0539 2171 7629 3176 7523 8467 4818 4676 6940 5132
       0005 6812 7145 2635 6082 7785 7713 4275 7789 6091
       7363 7178 7214 6844 0901 2249 5343 0146 5495 8537
       1050 7922 7968 9258 9235 4201 9956 1121 2902 1960
       8640 3441 8159 8136 2977 4771 3099 6051 8707 2113
       4999 9998 3729 7804 9951 0597 3173 2816 0963 1859
       5024 4594 5534 6908 3026 4252 2308 2533 4468 5035
       2619 3118 8171 0100 0313 7838 7528 8658 7533 2083
       8142 0617 1776 6914 7303 5982 5349 0428 7554 6873
       1159 5628 6388 2353 7875 9375 1957 7818 5778 0532
       1712 2680 6613 0019 2787 6611 1959 0921 6420 1989

A>
```

Figure-12.1.8〝PI1000.COM〞の実行

このようにプログラムの開始時刻が表示された後, しばらく沈黙し, がんばって働いているのか, 暴走でもして, どこかへ行ってしまっているのか不安な時が流れますが大丈夫, いたって真白目に仕事をしていたようで, 演算の終了時刻を表示した後, 1000桁をプリントアウトして, CP／M にもどりました.

所要時間は「52分27秒」でした. 誰にも書ける BASIC 言語による短いプログラムで, πの1000桁を

1時間以内に 8bit のパーソナル・コンピュータで行うことが可能なのです．このことを読者はどう評価されるでしょう．19世紀の終りには，526桁まで正しく求められていたと本には書かれています．実に紙とペンだけで．人々のπへの挑戦の歴史を振り返り，机の上にチョコンと置いてある小さなコンピュータを眺めると，いろいろ考えさせられます．

12.2 MBASICインタプリタによる実行

PC-8001，if800，FM-8など多くのパーソナル・コンピュータで採用されているマイクロソフト社の BASIC のオリジナル版が，この BASIC-80であり，そのプログラム名が MBASIC なのです．

MBASIC の起動と，先ほど作成したソース・プログラム "PI1000.BAS" をロードして，LIST で確認するまでのサンプルランを Fig-12.2.1に示します．MBASIC が起動して，"OK" のプロンプトが出れば後は通常の BASIC と同じです．

```
A>MBASIC↲

BASIC-80 Rev. 5.2
[CP/M Version]
Copyright 1977, 78, 79, 80 (C) by Microsoft
Created: 14-Jul-80
22598 Bytes free
Ok
LOAD "PI1000"↲
Ok
LIST↲
100 '       ----------------------------------------------------
110 '       =====  CP/M Learning System -I-  DEMO Program  =====
120 '       =====                                          =====
130 '       =====              PI 1000 Figures             =====
140 '       ----------------------------------------------------
150 '
160 DEFINT A-Z: DEFSNG R,M,X
170 DIM A(252),B(252),C(252),D(4),F(4)
180 PRINT: PRINT "START TIME = "; CHR$(&H1B); "A"
190 GOTO 260
200 GOSUB 210: FOR I=1 TO 252: A(I)=B(I): NEXT: FOR J=1 TO 252: B(J)=0: NEXT: RE
TURN
       .
       .
       .
       .
520 PRINT "7";: GOTO 550
530 PRINT "8";: GOTO 550
540 PRINT "9";
550 NEXT: RETURN
560 '
570 'END OF LIST
Ok
```

Figure-12.2.1 MBASICでプログラムを作る

さあこれで RUN すれば実行が開始されますが，以前 PC-8001の NBASIC で行った時に，5時間以上もかかっていますので今回は RUN した後は安らかに眠ることにして，明日の朝のお楽しみということにしましょう．これも if800 CP／M 上で実行します．

そのサンプルランを Fig-12.2.2に示します．

```
RUN
START TIME =  81/08/04 TUE 00:00:08

END TIME   =  81/08/04 TUE 03:59:03

PI= 3. 1415 9265 3589 7932 3846 2643 3832 7950 2884 1971
       6939 9375 1058 2097 4944 5923 0781 6406 2862 0899
       8628 0348 2534 2117 0679 8214 8086 5132 8230 6647
       0938 4460 9550 5822 3172 5359 4081 2848 1117 4502
       8410 2701 9385 2110 5559 6446 2294 8954 9303 8196
       4428 8109 7566 5933 4461 2847 5648 2337 8678 3165
       2712 0190 9145 6485 6692 3460 3486 1045 4326 6482
       1339 3607 2602 4914 1273 7245 8700 6606 3155 8817
       4881 5209 2096 2829 2540 9171 5364 3678 9259 0360
       0113 3053 0548 8204 6652 1384 1469 5194 1511 6094
       3305 7270 3657 5959 1953 0921 8611 7381 9326 1179
       3105 1185 4807 4462 3799 6274 9567 3518 8575 2724
       8912 2793 8183 0119 4912 9833 6733 6244 0656 6430
       8602 1394 9463 9522 4737 1907 0217 9860 9437 0277
       0539 2171 7629 3176 7523 8467 4818 4676 6940 5132
       0005 6812 7145 2635 6082 7785 7713 4275 7789 6091
       7363 7178 7214 6844 0901 2249 5343 0146 5495 8537
       1050 7922 7968 9258 9235 4201 9956 1121 2902 1960
       8640 3441 8159 8136 2977 4771 3099 6051 8707 2113
       4999 9998 3729 7804 9951 0597 3173 2816 0963 1859
       5024 4594 5534 6908 3026 4252 2308 2533 4468 5035
       2619 3118 8171 0100 0313 7838 7528 8658 7533 2083
       8142 0617 1776 6914 7303 5982 5349 0428 7554 6873
       1159 5628 6388 2353 7875 9375 1957 7818 5778 0532
       1712 2680 6613 0019 2787 6611 1959 0921 6420 1989

OK
```

Figure-12.2.2 MBASICでの実行

このようにインタプリタではかろうじて4時間を切り「3時間58分55秒」で終り，結果は同様にプリントアウトされています．夜中の12時に RUN して，夏のことですから東の空が白む早朝の4時頃に長かった演算を終えたわけです．

12.3 MBASICとコンパイラの比較

コンパイラを使用するメリットは，高速化を計るだけではありません．第1の理由は，インタプリタが不用になると言うことです．つまり，あるプログラムを実行するのに，BASIC 自身は必要でなくなると言うことです．今回の例では，コンパイルによって最終的に得られた，πを1000桁求めるプログラム〝PI1000.COM〟は，純マシン語のプログラムであり，MBASIC とは何ら関係なく，〝PI1000.COM〟自身で単独に働くものなのです．又，ROM上で働くプログラムを作ることも可能です．

そしてこのプログラムは，メディア変換（3.2章参照）することによりそっくりそのまま，あらゆるCP／M マシンで実行することができるのです．

高速化については，もっと十分に検討すれば，さらに速くなると思います．高速化について1つの実験をしてみましょう．簡単なプログラムですが，配列A（J）を1万個確保し，その1万個を0クリアする作業を10回くり返す（計10万回0クリアする）と言うプログラムを実行して，その所要時間を計ります．

プログラム･リストを Fig-12.3.1に，MBASIC インタプリタでの実行を Fig-12.3.2に，それをコンパイルしたもの（プログラム･ファイル名〝FORNEXT.COM〟）での実行を Fig-12.3.3に示します．

```
100 DEFINT A-Z
110 DIM A(10000)
120 PRINT "START = "; CHR$(&H1B); "A" ' --- TIME DISPLAY
130    FOR I=1 TO 10
140          FOR J=1 TO 10000
150             A(J)=0
160          NEXT J
170    NEXT I
180 PRINT: PRINT "END   = "; CHR$(&H1B); "A" ' --- TIME DISPLAY
190 END
```

Figure-12.3.1 サンプル・プログラム

```
RUN↲
START =   81/08/04 TUE 20:10:05

END   =   81/08/04 TUE 20:16:34
OK
```

Figure-12.3.2 MBASICによる実行

```
A>FORNEXT↲
START =   81/08/04 TUE 20:20:04

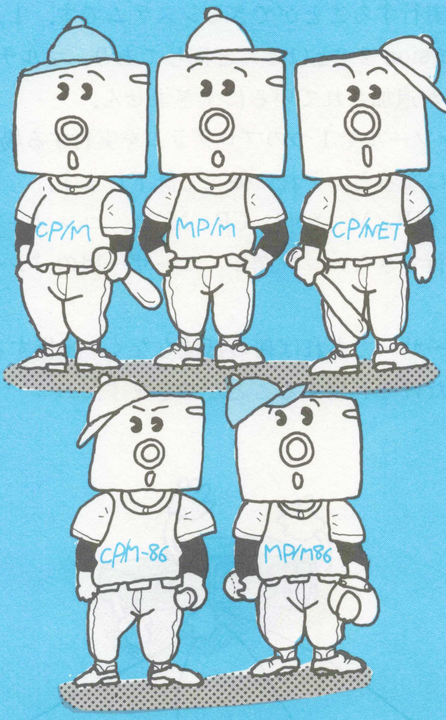
END   =   81/08/04 TUE 20:20:10

A>
```

Figure-12.3.3 BASICコンパイラによる実行

これらの実行も，時刻を表示させるに便利な if800の CP／M で行いました．このように整数型の配列の扱いではコンパイラは何んと60倍も速くなっています．このような特徴をよく知って，それなりのプログラミングをすれば，BASIC コンパイラは大変強力な言語として，十分に各方面で利用できるでしょう．

13章　CP/Mファミリー



MP/M
CP/NET
CP/M-86, MP/M-86

みなさんも,MP／M とか CP／NET などの言葉を,どこかで見たり聞いたりしていることと思います.何んとなく CP／M と関係がありそうな言葉ですが,その通りで CP／M の延長線上にあるものなのです.それらの概要をここで紹介しておきましょう.

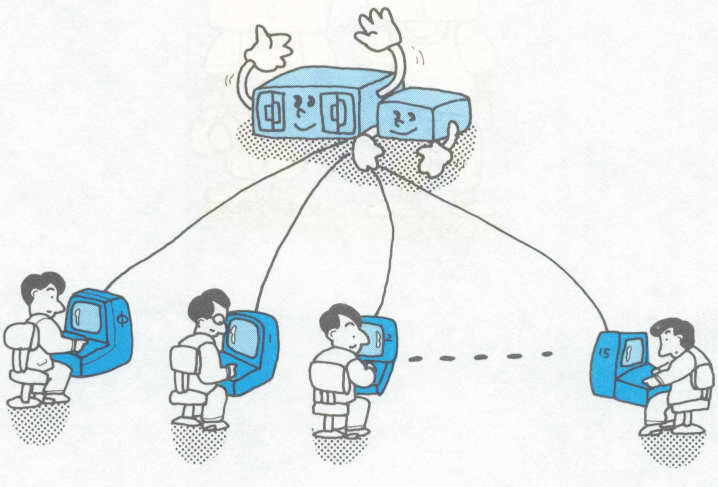
13.1 MP/M

MP／M (Multi programming／Monitor)は,リアルタイム・マルチプログラミング用の OS であり,1台のコンピュータで1～16台のコンソールが接続可能であり,同時に複数のユーザーが独自のプログラムを8プロセスまで実行することができるシステムです.1人だけで使用する時は,使い方やコマンドは,CP／M の version 2.2とほとんど同じであり,マルチ・ユーザーで使用する時もいくつかの MP／M用のコマンドが追加されているにすぎません.

処理スピードは,1台のコンソールで1つのプログラムを実行する場合は,CP／M と同程度ですが,コンソールやプロセスが複数になると,それに従って低下します.

このマルチ・ユーザー用の MP／M に対し CP／M は,コンソールは1台のみ接続可能で,同時には1つのプログラムしか実行できないシングル・ユーザー用の〝シーケンシャル・システム〟であるわけです.

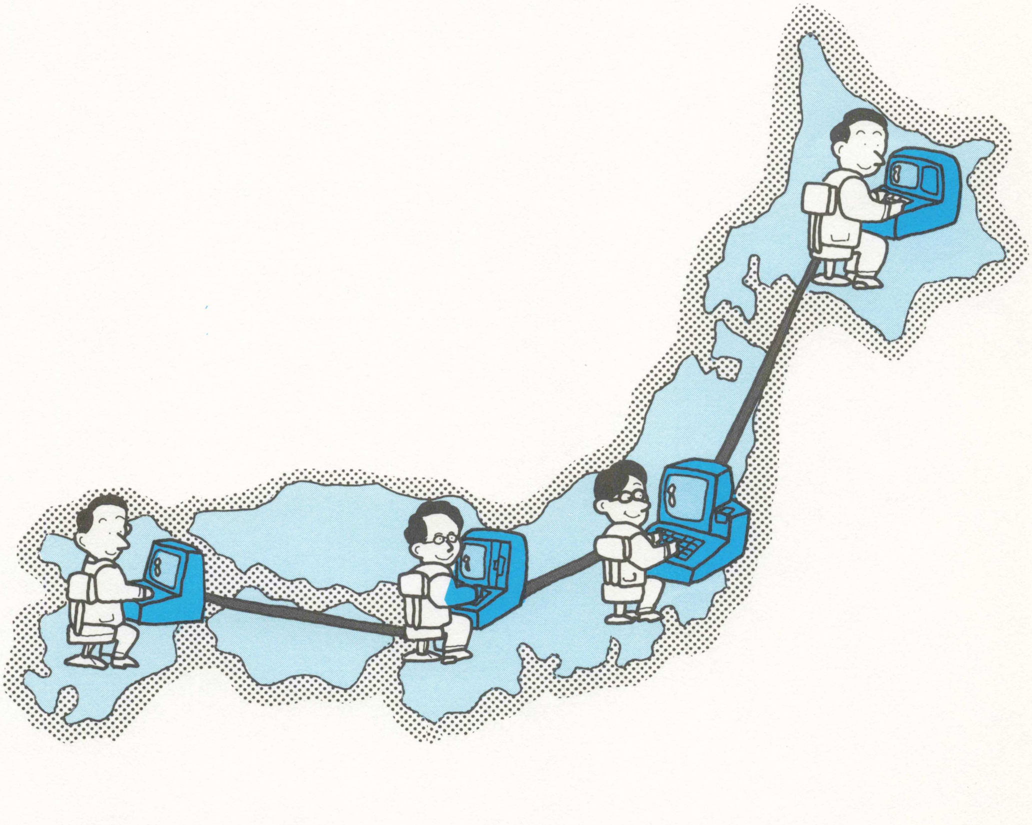
MP／M は,CP／M のユーザーであれば何の抵抗もなく,操作することができ,あとは MP／M 独自のコマンドを数種覚えるだけで利用することができます.



13.2 CP/NET

CP／NET (Control Program／for a microcomputer NETwork) は，MP／M システムをマスターとして，CP／M または MP／M をスレーブとして，通信回線などを介してネットワークを組む，ネットワーク・オペレーティング・システムであり，互いのディスク・ファイルの共用，コンソールやプリンタ，プログラムおよびデータ・ベースの共同利用を目的とした Digital Research の新しいシステムです．

マスターとスレーブのネットワークは，いろいろな構成をとることができ，例えば東京のソフトウェアを大阪でもらって実行し，結果のデータを福岡に送ったりすることなど，いろいろな使い方が可能です．



13.3 CP/M-86, MP/M-86

16ビットの8086プロセッサ用の CP／M と MP／M であり，CP／M-86はすでにいくつかのパーソナル・コンピュータに採用されていますし，MP／M-86については，1981年の8月にリリースされるとアナウンスされています．いずれも CP／M のversion 2.0以上と同じファイル・システムを使用しており，メモリ空間が拡張されたための付加機能がある他は，8bit の CP／M，MP／Mと同様の使い方で16bit の8086のコントロールを行うことができるものです．

16bit のチップは何が主流になるのか関心が持たれていますが，これも8bit で8080や Z80が圧到的な地位を占めたのと同じく，周辺のソフトウェアがどれだけ充実されるかに掛っていますが，内外とも CP／M-86マシンが続々と発表されている現状から，どうやら8086が主流になるとの見方が多いようです．

あとがき

以上が CP／M の入門編ですがいかがですか？　BASIC 言語の沢山のコマンドやステートメントなどを覚えるのに比べれば，まだやさしいかな？っと思いますが…….

本書は私が CP／M を始めた頃，Digital Research の英文マニュアルしか参考書がなく，ここも分らない．あそこも分らない，こうしたいんだけど，どうすればいいのか……などと試行錯誤をしたことを思い出しながら，〝あの頃こんな本があったら楽だったのに〟という思いで構成したものです．

入門編として，まだまだ解説しなければならないことがあると思いますが至らない点はご容赦下さい．

本書に続いて，〝実習 CP／M〟では CP／M の全コマンドのほとんどすべての使い方について，徹底実例解説を行い，〝応用 CP／M〟では 応用編として，さらに深く解説を行います．

XEROX，ヒューレット・パッカード，CDCなどの超有力メーカーの CP／M 参入は，国際的に一段と CP／M 指向を加速させることでしょう．

本シリーズを書くために，日頃お世話になっている方に感謝致します．

Digital Research 社の総代理店でもある，マイクロソフトウェア・アソシエイツの社長でコンピュータ関連に造詣が深い岡田純一氏には，本シリーズのために心良くソフトウェアの提供をしていただきました．そして，シンセサイザーの神谷重徳氏や，Rgy Co. の片桐明氏などにも，いつも教えていただくばかりです．

付録A　CP/M上で走るソフトウェア

現在国内でいつでも入手可能な一般的なソフトウェアを，分野別に紹介します．ソフトウェアを購入する場合，8インチ標準フロッピー・ディスク以外のシステムでも，使用するCP／Mマシンは何かを指定すれば，たいていの機種用にメディア変換されたディスケットが用意されていますので即実行させることができます．また，マニュアルに関しては和文が用意されているものもあります．しかし別ルートで輸入されたものは和文マニュアルが付かない場合もありますので，和文が必要な方はあらかじめ確認するとよいでしょう．

エディタ，アセンブラ

製品名	会社名	内容
EDIT-80	Microsoft	行単位のランダム・アクセスによるテキスト・エディタ．編集機能のほとんどを備えており，かつ高速．行単位が，MACRO-80，FORTRAN-80のリストと同じ．
WORDMASTER	Micro Pro	カーソル・アドレッシング可能なCRTターミナルではスクリーン・エディタが行える．通常のポインタ形式のエディタ・モードではCP／MのEDとコマンドがコンパチブル．
MAC	Digital Research	8080のインテル・マクロ・アセンブラの上位コンパチブルに当るマクロ・アセンブラ．各種ライブラリを含み，Z80のライブラリによりZ80もアセンブル可能．SIDで用いられるシンボル・ファイルも作成される．
MACRO-80	Microsoft	インテル標準のマクロ機能を含み，高速かつ強力な8080／Z80のマクロ・アセンブラ．Z80のニーモニック・オペレーション・コードもアセンブル可能．LINK-80リンキングローダなども含まれている．

言語プロセッサ

製　品　名	会　社　名	内　　　　容
BASIC-80	Microsoft	ANSI BASICのサブセット・スタンダード．変数名40字まで使用可．ランダム・ファイル時の可変長レコード．WHILE／WEND条件文．CHAIN／COMMONによる結合と変数共用など．
BASIC COMPILER	Microsoft	1パス方式で最適化された8080マシン語コードを出力する．ソース・プログラムはBASIC-80とコンパチブル．実行速度はBASIC-80の3～10倍．COBOL-80，FORTRAN-80，MACRO-80などによるリロケータブル・オブジェクト・ファイルとリンク可能など．
COBOL-80	Microsoft	ANSI-74を満足．さらに他のプログラムへ移行のCHAIN，CRTターミナルのためのACEPT／DISPLAY，トレース方式のデバッグ機能，その他が拡張されている．MACRO-80と同じリロケータブル・オブジェクト・コードを生成するなど．
CIS-COBOL	Micro Focus	ANSI-74に準拠．インテル社がOEM向けに販売しているiCIS-COBOLとして供給しているもの．64KCP／Mでは8000ライン以上のソース・ステートメントをコンパイル可能．マイクロコンピュータで最強のCOBOLと言われている．強力なスクリーン・ハンドリング可能．
FORTRAN-80	Microsoft	ANSI-66に準拠．さらにプラス127～マイナス128のLOGICAL変数．高速なLOGICAL DOループ．その他が拡張されている．1パス方式のコンパイラで，最適化されたリロケータブル・オブジェクト・コードを生成するなど．
PASCAL／M	Sorcim	コンパイラはPコードを生成し，PコードはPコード・インタプリタで実行される．ランダム・ファイル・アクセス，実行中のデバック・サポート，CASE文におけるOTHERWISE，32ビット9桁演算など．
PL／I-80	Digital Research	ANSIサブセットGに準拠．DATE GENERAL社，PRIME COMPUTER社のPL／Iと完全コンパチブル．IBM社のフルセットPL／Iと上位コンパチブル．3パス方式のコンパイラで，MACRO-80と同じリロケータブル・オブジェクト・コードを生成する．
mu SIMP mu MATH-80	Microsoft	SIMP＝Structured IMPlementation language．記号処理言語プロセッサ．MATH＝Symbolic Math Package．数学パッケージ．数学・科学教育，工学分野のプログラミングに最適．各パッケージはそれぞれmuSIMP，muMATHで書かれており，その言語自身も含まれている．

製　品　名	会　社　名	内　　　　　容
FORTH／SL 5	Stackworks	独自のアセンブラやデバッガが付属しており，マシン語とのインターフェイスが容易. 生成されるオブジェクト・コードは，RAM 上およびROM上での使用に合わせた，それぞれのターゲット・コンパイルが可能.
C-Compiler	White Smith	UNIX を記述した言語の UNIX version 7C に準拠. MACRO-80によりリロケータブル・オブジェクト・ファイルとリンク可能.
mu LISP-80	Microsoft	LISP 1.5に準拠. LISPインタプリタ. 実用的なリカーシブ言語.

ユーティリティー

製　品　名	会　社　名	内　　　　　容
DESPOOL	Digital Research	バック・グラウンドでプリント・アウトを行わせて，同時に別のプログラムを実行するためのプログラム.
SID ZSID	Digital Research	8080 およびZ80用のシンボリック・デバッガ. MACによって作られたシンボル・データを使って強力な参照が可能. シンボル演算式を受け付ける. パス・カウント機能. ローカル・トレース，ROMへのトレース機能など.
TEX	Digital Research	プリント・アウト用のテキスト・フォーマッタ. ページNo.，マージン，ページ・ヘッダー，などのフォーマッティング処理をしてプリンタへ送る.
Utility Vol. 1	Syncware Labs	ユーティリティー・プログラム7種のパッケージ. ディスク間コピー，2つのディスクの比較，2つのファイルの比較，セクタ・ディスプレイ，メモリ・テストなど. 比較プログラムは非常に強力.
SUPERSORT	Micro Pro	データ・ファイルのランダムな情報を選別して，並べ変えたりするソート・プログラム.

アプリケーション

製品名	会社名	内容
WORDSTAR	Micro Pro	CP/M上で動作する現在で最も強力・多機能なワード・プロセッサ(英文). ワード・プロセッサに要求される機能のほとんどを有している. カーソルのアドレッシング可能なターミナルを使用して, スクリーン上で作業を行う.
MAILMERGE	Micro Pro	WORDSTARと併用すると, チェイン・プリント, ネステッド・プリントなど2～3のファイルからデータをマージすることが可能. データ・ファイルから変数で呼出しを行い, テキストへ挿入が可能.
DATASTAR	Micro Pro	データ・エントリーシステム, インプット・フィールドの照合は, 文字数, マスク, アトリビュートにより行う. データ・ファイルはCP/M上の言語とコンパチブル.

クロスソフトウェア

製品名	会社名	内容
XMACRO-86	Microsoft	8080システム上で8086用のコードをアセンブルする. 1分間に1000行以上の高速アセンブルを行う. MACRO-80と同等の機能を持つ.
TRANS-86	Sorcim	8080/Z80から8086/88へのプログラム・トランスレーター. アセンブリ言語レベルでのトランスレーションを行い, A. C. T. IIクロス・アセンブラにより実行モジュール・コードを生成できる.
A. C. T.	Sorcim	汎用クロス・アセンブラ. 現在サポートされているマイクロプロセッサは, Z80, 8080, 8085, 6502, 680x, 8048, 8049, 8050, 8085, 8041, 8035, 8086/88, Z8000, 68000. CP/M上でこれら任意のプロセッサのマシンコードを生成する.
XS-8000	Rgy Co	Z8002のアセンブリ・ソースをクロスアセンブルし, オブジェクト・ファイル, リスト・ファイルを出力する. 書式はZilog社のPLZ/ASMアセンブラとコンパチブル.

以上のソフトウェアはマイクロソフトウェア・アソシエイツ(TEL03-497-0381)の取扱いカタログに載っているもので, お問合せはそちらへどうぞ.

その他にも, 言語では APL, ALGOL や, PL/M とコンパチブルの PLMX なども国内で入手でき, アプリケーション・ソフトなども数多く入手できます.

付録B　国産CP/Mマシンリスト

みなさんも雑誌の広告などを〝CP／M〃という文字に注意してパラパラと御覧になればお分かりのように，実にたくさんの機種でCP／Mが走っています．ここでは，これらの中から代表的なもののいくつかをリストアップしました．また現在CP／Mマシンを計画中のメーカーは大手中小とも数多くあり，CP／Mが走る新しい機種は続々と増え続けています．

まえがきでも述べましたが，アメリカの情況はよほどの低価格のものでない限り，CP／Mが走らなければ市場では相手にされないと言っても過言ではありません．それらのうちで日本で容易に入手できる，代表的な機種も数機種付け加えてあります．また，XEROX，ヒューレット・パッカード，CDCなどのCP／Mマシンも，間もなく日本で入手できることでしょう．

CP／Mの走る国産マシン一覧表

メーカー名(順不同)	機　種　名	摘　　要
NEC	PC-8001	
〃	PDA-800	
シャープ	MZ-80 K／C	
〃	MZ-80B	
沖電気工業	if 800	
アドテック	IDS-9000Z	
東京三洋電機	MBC-2000／7	
〃	FDS-1000	
東洋通信工業	AVC-777	
大阪ICM	FD 8080 システム	PC-8001と組み合わせ 8インチ・ディスク・システム
National	my brain シリーズ	
電　産	DSC-80 ZA	
国際データ機器	PDS-V	
ザックス	ZDS-8000	
ロジック・システムズ・インターナショナル	iBEX 7000	CP/M-86
Y・Eデータ	YD-8100	
富 士 通	MICRO-8	要Z80カード
中央電子	CEC300, CEC500, CEC800	

国産CP/Mマシン一覧表

メーカー名(順不同)	機　種　名	摘　　要
アイ電子測器 東芝 工人舎 エプソン	FDPS パソピア BLACK BOX QC-20	
タンディ・ラジオ・シャック Apple North Star クロメムコ	TRS model II Apple II HORIZON クロメムコ	 要Z80カード

※この他にも，CP/Mの走る機種はあります．また，これからも続々登場することでしょう．

索　引

ア

カ

サ

タ

ナ

ハ

マ

ヤ

ラ

著者とＣＰ／Ｍの出会い

村瀬　康治

マイクロ・コンピュータとの付き合いは，歴史的存在のＴＫ-80（ＮＥＣ）＋ＴＶＤ-01（アドテック）の時代から．

当時は，秋葉原の本屋でやっと手に入るkilobaud誌などを読みあさり，8080系とＳ-100バスこそ本命！などと思い，知る人ぞ知る往年の名機，ＳＯＬ-20を自分で輸入し，その高度な周辺ソフトウェアに驚くと共に，8080のソフトウェアを知るよい材料になった．

さらに，その頃アメリカで既に普及の域に達していたＣＰ／Ｍの存在を知り，Tarbell社のディスク・コントローラを組み立て，ＣＰ／Ｍをインプリメントした．

その後，我国で最初のまとまった記事であった，月刊アスキー1979年５月号からの「How To CP／M」を連載し，この頃から「ＣＰ／Ｍの本，書かなくちゃいけないネ.」と思っていた．

1981年５月，これも本邦初の「標準ＣＰ／Ｍハンドブック」を監訳，今ベストセラーとなっている．そして思い始めて２年余，”分かり易い”本書の出版となる．

テレビ朝日技術局勤務
初版時　35才

入門CP/M
アスキー・ラーニングシステム①入門コース

1981年10月20日　初版発行
1984年3月5日　第1版16刷発行
定価1,500円

著　者　村瀬　康治
発行者　塚本慶一郎
発行所　株式会社アスキー
　〒107　港区南青山5－11－5住友南青山ビル5F
振　替　東京7-57496
電　話　03－486－7111(代表)

編集担当者　井芹昌信
印刷　壮光舎印刷

ISBN4-87148-600-1 C3055 ¥1500E

アスキー・ラーニングシステム
ALS
ASCII Learning System
①入門コース

定価1,500円

アスキー出版局

ISBN4-87148-600-1 C3055 ¥1500E